NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

COORDINATED INLAND AREA SEARCH AND RESCUE (SAR) PLANNING AND EXECUTION TOOL

by
Timothy S. Castle

September 1998

Thesis Advisor: Gordon Bradley
Thesis Advisor: Alan Washburn
Second Reader: James Eagle

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
September 1998

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
COORDINATED INLAND AREA SEARCH AND RESCUE (SAR) PLANNING AND EXECUTION TOOL

5. FUNDING NUMBERS
N000149ΩP20001

6. AUTHOR(S)
Castle, Timothy S.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research, 110 Duncan Avenue,
Suite 100, Bolling AFB, DC 20332-0001
Office of Naval Research, 800 North Quincy Street
Arlington, VA 22217

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense, the Department of Transportation, or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis designs and implements the Coordinated Inland Area Search and Rescue (SAR) System (COINSS). This system provides several important features not provided by current inland SAR computer systems. First is the ability to model movement of the target. Second is modeling the effect terrain has on the movement of the target. Third is the visual presentation of a probability map, a color display showing the probability that the target is located at various geographic positions. COINSS is developed in the Java programming language. It is designed to be implemented with a map-based planning system using loosely coupled components. COINSS provides the initialization, movement, and search algorithms which are used by the planning system to support the search operation. The initialization algorithms define the search area where the SAR operation will occur. Initial areas are defined for the target. COINSS models the movement of the target as a discrete time Markov chain. Bayes theorem is used to update the probability map when negative search information is provided. This thesis will improve inland SAR operations by providing the first model with an interactive graphical user interface and a model of target movement.

14. SUBJECT TERMS
Search and Rescue, Java, Loosely Coupled Components, Map Based Planning

15. NUMBER OF PAGES
99

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFI- CATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
UL

Approved for public release; distribution is unlimited

COORDINATED INLAND AREA SEARCH AND RESCUE (SAR) PLANNING AND EXECUTION TOOL

Timothy S. Castle
Lieutenant, United States Coast Guard
B.S., United States Coast Guard Academy, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1998

Author: ________________________
Timothy S. Castle

Approved by: ________________________
Gordon H. Bradley, Thesis Advisor

____
Alan R. Washburn, Thesis Advisor

____
James Eagle, Second Reader

Richard E. Rosenthal
____
Richard Rosenthal, Chairman
Department of Operations Research

ABSTRACT

This thesis designs and implements the Coordinated Inland Area Search and Rescue (SAR) System (COINSS). This [illegible]n provides several important features not provided by current inland SAR computer systems. First is the ability to model movement of the target. Second is modeling the effect terrain has on the movement of the target. Third is the visual presentation of a probability map, a color display showing the probability that the target is located at various geographic positions. COINSS is developed in the Java programming language. It is designed to be implemented with a map-based planning system using loosely coupled components. COINSS provides the initialization, movement, and search algorithms which are used by the planning system to support the search operation. The initialization algorithms define the search area where the SAR operation will occur. Initial areas are defined for the target. COINSS models the movement of the target as a discrete time Markov chain. Bayes theorem is used to update the probability map when negative search information is provided. This thesis will improve inland SAR operations by providing the first model with an interactive graphical user interface and a model of target movement.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The National Search and Rescue P establishes the United States Air Force as the executive agent for Inland Area Sear d Rescue (SAR). This area encompasses the Continental United States, except Alaska, and waters under the jurisdiction of the United States. The Air Force established the Air Force Rescue Coordination Center (AFRCC) as the contact point for federal resources for inland SAR [18].

In addition, the National Search and Rescue Plan dictates that Search and Rescue Coordinators should provide for the fullest cooperation and use of additional Federal military, civil, State, local, and private agencies as may be necessary and practicable [18]. The On-Scene Commander is responsible for overseeing the execution of the entire search plan. The Commander uses search tables to formulate each assignment in the overall search plan. The individual search assignments are then provided to each search unit for implementation. The results of the individual search assignments are communicated back to the Commander and used to update the search scenario. Based upon the updated scenario, the Commander reevaluates the search plan, creates new search assignments, and the process is repeated.

Currently, numerous diverse software tools and spreadsheets are available to help do numeric planning calculations and table referencing. These stand-alone aids compute the data for the Commander, who must then interpret the data and incorporate it into the search plan. Additionally, management programs are available for ongoing analysis of the search plan.

However, despite the availability of a variety of search software programs, three key factors prevent their widespread use and limit their usefulness during an active inland search operation. These factors are:

1. the lack of a graphical mapping interface,
2. the lack of inter-operability among the computer systems of cooperating agencies, and
3. the inability to account for a mobile target.

This thesis designs and implements the Coordinated Inland Area Search and Rescue (SAR) System (COINSS). This system accounts for the three factors not provided by current inland SAR computer systems. First, COINSS provides a color display of the target probability distribution, which can be used as an overlay on a map-based planning system. Second, it is developed in the Java programming language and is designed to be implemented with a map-based planning system using loosely coupled components. Finally, it models the movement of the target as a discrete time Markov chain.

COINSS provides the initialization and search algorithms which are used by the planning system to support the search operation. The initialization algorithms define the search area where the SAR operation will occur. Initial areas are defined for the target. Bayes theorem is used to update the probability map when negative search information is provided.

This model will improve inland SAR operations by providing the first model with an interactive graphical user interface and a model of target movement.

ACKNOWLEDGEMENT

I would like to thank Professor Gordon H. Bradley and Professor Alan R. Washburn for their guidance and assista the development and completion of this project. Professor Bradley initiated my interest in the topic of creating a SAR model which would promote interagency cooperation. Professor Washburn helped to focus this model on the inland search environment, where more interagency cooperation is required. Both individuals provided extensive insight and direction in completing this thesis.

I would also like to thank the contributions of the "Loosely Coupled Components" research group at NPS. This includes Professor Arnie Buss, Major Jack Jackson, and Captain Allan Bilyeu. Professor Buss and Major Jackson wrote the code for the map-based planning system which I used to incorporate my SAR system. Captain Bilyeu provided an initial implementation to the loosely coupled map-based planning system. This allowed my implementation with the same system to occur more smoothly.

Finally, I need to thank my wife, Sandy, and three daughters, Jaime, Elizabeth, and Carolyn. They each made large personal sacrifices during my work on this thesis. Without their love and support I could not have completed this project. I love you each very dearly.

THESIS DISCLAIMER

The reader is cautioned that the computer programs developed in this research may not have been exercised for all cas ıterest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

I. INTRODUCTION

A. BACKGROUND

1. The Responsibility of Lanc ı Search and Rescue (SAR)

The National Search and Rescue Plan establishes the United States Air Force as the executive agent for Inland Area Search and Rescue (SAR). This area encompasses the Continental United States, except Alaska, and waters under the jurisdiction of the United States. The Air Force established the Air Force Rescue Coordination Center (AFRCC) as the contact point for federal resources for inland SAR [18].

In addition, the National Search and Rescue Plan dictates that Search and Rescue (SAR) Coordinators should provide for the fullest practicable cooperation and use of additional Federal military, civil, State, local, and private agencies as may be necessary and practicable [18]. To this end, the AFRCC has established agreements with each State, which describe the responsibilities for different emergencies. However, these agreements are not consistent from State-to-State, and often do not fully address the availability of local and private agencies to assist with the search effort.

2. SAR Tools and Process

Search planning is a well-researched and documented discipline. The National Search and Rescue Plan provides a complete set of tables and charts for creating and calculating search areas and search patterns.

The On Scene Commander is responsible for overseeing the execution of the entire search plan. The Commander uses search tables to formulate each different search assignment in the overall search plan. The individual search assignments are then provided to each search unit for implementation. The results of the individual search assignments are communicated back to the Commander and used to update the search scenario. Based upon the updated scenario, the Commander reevaluates the search plan, creates new search assignments, and the process is repeated.

The bulk of search planning is still done on paper charts and maps. The Commander must take into account a large range of factors such as the target's last known position, time of that position, the terrain that is being searched, the type of target that is being sought, the number, experience, and composition of the available search units, etc. The Commander uses this information, as well as personal experience and all available charts and tables, to lay out the search area, divide the area into appropriate search sectors, assign individual search teams to cover the sectors, and establish the search pattern(s) that will be used by the search teams.

Currently, numerous diverse software tools and spreadsheets are available to help do numeric planning calculations and table referencing. These stand-alone aids compute the data for the Commander, who must then interpret the data and incorporate it into the search plan.

Additionally, management programs, such as CASIE III [3], are available for ongoing analysis of the search plan. These programs allow the searcher to define a search region and divide that region into search areas. Each area is assigned a probability that the target is located within that area. The searches are then conducted in the various search areas and the results used to update the probabilities by applying Bayes theorem.

An unsuccessful search in an area reduces the probability that the target is in that area. This reduction in probability is directly related to probability of success of the search method used.

B. PROBLEM

Despite the availability of a variety of search software programs, three key factors prevent their widespread use and limit their usefulness during an active inland search operation. These factors are:

1. the lack of a graphical mapping interface,
2. the lack of inter-operability among the computer systems of cooperating agencies, and
3. the inability to account for a mobile target.

CASIE III is an example of current software that provides the Commander with the raw data required to make search decisions. The program takes user input search areas and target location probabilities and the program provides, as output, probabilities that the target is located within the predefined search areas. When the results of a search are entered, the program then updates the probabilities, and outputs these revised values. The Commander is then responsible for translating that information to a physical map representation. Then the Commander assesses the search region and evaluates the next step to be taken in the search execution. The new search assignments, and eventually search results, must then be entered back into the program, where the cycle resumes.

The second factor that must be considered is the wide assortment of resources that can be brought to help execute a SAR case. The range of available personnel resources goes from trained, full-time federal and state professionals all the way to sparsely trained local volunteer organizations. The computer resources that will be available from these sources are unpredictable at best. In order for a system to be useful for all organizations participating in the search, the program must be able to run or be viewed on any computer system.

The third factor is the abilities of the program itself. No model currently exists which accounts for a moving target on land. Probabilities are assigned to the various areas of the search region. After these probabilities are assigned, the target is assumed to be immobile. The changes in the probabilities in each area are only the result of unsuccessful searches that were conducted.

C. APPROACH

This thesis develops the Coordinated Inland Area SAR System (COINSS) to provide one method of resolving the problems addressed above. The design of this computer program was based upon several key elements.

1. The Key Elements

a. Portable

Due to the urgent and unpredictable nature of SAR, it is not possible to prepare in advance for all possible scenarios. The Commander must make best use of the resources available at the time. The computer program must be small enough to be easily and quickly loaded from a disk or downloaded over a network and provide a user interface that requires little user training. Additionally, the program should be able to opera a wide variety of systems.

b. Graphical interface

A graphical interface would allow the Commander to do the SAR planning on the system, and not require a transfer of information between a paper chart and the computer model. The visual display of probabilities would provide a better description of the search progress. Additionally, the direct interface between the visual display and the search logic would provide a quicker update to the search execution once other search information has been received.

c. Interoperable with available maps

Again, SAR is unpredictable. We cannot know where the next SAR case will occur. The program should allow for use of maps from any electronic source to provide the graphical planning interface.

d. Allow for target movement

A target may not remain stationary. Ignoring the motion of the target in the execution of the SAR plan significantly impacts the location of the search areas, the search method used in those areas, and the results of the search.

e. Common picture view

One advantage of the portability element is that all units coordinating on a search would be able to view the planning tool to assist in their search execution. The Commander would maintain and update the operational view of the system, which would be distributed to each search unit.

2. Two Concepts of Movement

Two choices are available to describe the target movement through the search area. Wagner [19] explains one method as being represented by a bundle of simulated tracks that encompass the many possibilities for target initial position and movement. The number of necessary tracks depends upon the complexity of the target being modeled, and should be set to a sufficiently large number in order to cover the complete range of possible initial conditions. A probability is associated with each track. This is the probability that the target is following the respective track. Wagner further explains how the tracks are monitored and updated, based upon negative search results. These tracks move over a grid search area, and require the use of random number generation to determine the progressing course of the simulated tracks.

The second method is a pure stochastic representation of movement [19]. A grid is placed over the search area. Each grid square, $S_{x,y}$, contains a velocity array, $V_{x,y,i,j}$, which contain elements that are the probability that the target is located in the square, moving with velocity i in the east-west direction, and velocity j in the north-south direction. The total probability for each square is computed by:

$$P_{x,y} = \sum_{i,j} V_{x,y,i,j}$$

During time-based updates, the probabilities are moved from square to square, based upon their velocity, and then the velocity state is updated based upon a Markov chain. Bayes theorem is used to update the probability distributions when negative search information is received. This method requires no random number generation.

D. COINSS

COINSS is designed as a component to be used in conjunction with a map-based planning system. The map-based planning system must have certain capabilities and provide distinct information to COINSS. First, the map-based planning system must be able to provide positions on the map by latitude and longitude. This is required when initially creating the search region, as well as when initial areas and search areas are created. Second, it must also be able to provide the number of pixels, both horizontally and vertically, between two points on the map. This is required when initially creating the search region. Finally, it must be able to receive numerical inputs that represent probabilities. This is required for initial area probabilities and search area probability.

COINSS is written in the Java programming language and is platform independent. This means that compiled code can be executed on a variety of computer systems that have the Java Virtual Machine (JVM), for example, Windows 95, 98, NT, and various UNIX platforms. COINSS can be loaded to any computer with the JVM and executed immediately. The map-based planning system used to demonstrate COINSS is also written in Java. Therefore, the total system can be distributed to execute on any computer with a JVM.

COINSS is written as a Java application, which means that it is executed from a command line prompt. COINSS and the map-based system could be converted to a Java applet, which would then be executed within a web browser that has a JVM, for example, Netscape Navigator or Internet Explorer. This would be the most convenient method of distributing a dynamic view of the search. This would allow searchers, family members, and others to have the same view of the search that the Commander has.

COINSS uses the stochastic representation for target movement, which is explained in complete detail in Chapter III. COINSS and a map-based planning system are available by contacting Professor Gordon Bradley at NPS, Monterey, CA 93943 or email to bradley@nps.navy.mil.

E. THESIS STRUCTURE

Chapter II explains the search and rescue process and discusses how the On Scene Commander would use a search movement model in a search and rescue operation.

Chapter III explains the logical search and movement elements of the Coordinated Inland area SAR System (COINSS).

Chapter IV explains the graphical implementation of COINSS.

Chapter V provides the conclusions and recommendations.

Appendix A provides the Markov transition matrices for the five classes of targets defined in COINSS.

Appendix B provides a walkthrough of COINSS operating with a map-based planning system.

Appendix C provides the code for SARgrid, and SARhex. This is the complete code for COINSS.

II. THE SEARCH AND RESCUE PROCESS

A. INITIAL ACTION

1. Obtaining an Initial Position

The initial area is the last known geographical location of the target. The initial area can be multiple disjoint or overlapping locations, in which case a probability is assigned to each portion sequentially, corresponding to the probability that the target is located in that portion. One of three general situations for the initial area information usually exists [18].

The first is a known position. This position can come from a direct witness, or may even come from the target individual. The single position is used as the initial area.

The second is a known track. In this instance the target's route of travel is known. What is unknown is the position along that route of travel. The complete length of the known track is used as the initial area.

The third is a known general area. This is the least favorable condition, but also the most likely. This instance occurs when general information regarding the habits of the target is known, such as where he typically hunts, or the part of the lake where she typically fishes. Natural boundaries, such as cliffs or rivers, can help to constrain this type of initial area.

In each case, the initial areas are assigned a probability, which is the probability that the target is located within that particular geographic area. The combined set of locations creates the initial area for the search.

2. Defining the Search Region

The total search region is the geographic area determined to be the most likely to contain the target. The bounds of this region need to account for the error in the initial position, the movement capabilities of the target, the amount of time the target has had to move, and the physical geographic constraints of the area [18].

The search region must be large enough to ensure that the target is located within it. Conversely, the total search region area should be small to minimize the probability of searching areas where the target is not located. Ideally, the total search region should be dynamically adjusted, to account for the both the target's movement, and the search efforts conducted.

If available, information about the elevation and terrain within the search region can be used to help predict the movement of the target.

B. SEARCH ACTION

The search action is a looping process. The process evaluates the current probability information, identifies the search areas, assigns units to conduct the search, executes the search, and then updates the probability information based on the new search information. This process is continually repeated, often for a variety of search units, until either the target is located or the search is suspended. Because this thesis does not focus on the tactical decisions required to create search plans, this section will only discuss the role that the outcome of those decisions has on the search movement model.

1. Identifying the Search Area

A search area is a sub-section of the search region that is searched by a particular search unit. Search areas are chosen to make best use of the abilities of the search unit in order to maximize the probability of detecting the target. Generally, the areas with a higher probability of containing the target are searched before the lower probability areas.

2. Search Execution and Results

After being assigned a search area, the unit searches that area. Once a search unit has completed its search, the unit reports its findings. If the target was found, the search can be concluded. If the target was not found, that information is used to update the probability map by Bayes Theorem. The probability that the target is in the searched area is reduced, and the probability that the target is in any non-searched area is increased. The change in probability on the probability map is directly related to the search unit's probability of detection of the target, given that the target is located within the area searched.

Upon completion of the search result update, a decision is made whether to continue the search or suspend the search. This decision is an easy one when the target has been located. When the target has not been located, the decision takes into account many factors. These factors include the length of time the search has already expended, the probability the target will be located given that it has not been found up to this time, and the value of the target. If it is decided to continue, the search cycle is repeated.

III. LOGICAL MODEL

A. PROBABILITY MAP COMPONENTS

The first layer in the logical model is the probability map. This map presents the probability that the target is located at a certain geographic position. This probability follows the laws of any standard probability distribution, and computations (such as Bayesian updates and using Markov chains) can be applied to it. It is this property which allows us to update the map for both time based movement and search result information [19].

1. Elements of the Grid Square

The logical map can be imagined as a large grid of squares. Each square is an individual piece that contains information relating to the complete map. Though the information values assigned to each square will vary from square to square, the type of information provided will be the same. As such, every square can be represented as an individual object, each having its own set of variables. The essential variables for each square are its position, elevation, and a velocity array. The first two of these variables are static. These values are set when the model is initialized, and will not change during the course of the search.

The velocity matrix, **V**, is a four-dimensional array, with the first two dimensions representing the location on the probability map, and the second two dimensions being speed in the east-west direction and speed in the north-south direction as it would appear on a common geographic map. The units for speed are feet per second. The values of the indices indicate the velocity in that direction. For example, $V_{x,y,4,3}$ represents the probability that the target is located in square $S_{x,y}$ moving with a velocity of 4 feet per second toward the east and 3 feet per second toward the north. Likewise, $V_{x,y,-4,-3}$ represents the probability that the target is located in square $S_{x,y}$ moving with a velocity of 4 feet per second toward the west and 3 feet per second toward the south.

B. INITIALIZING THE PROBABILITY MAP

There are two steps in initializing the logical probability map. The first step is to define the entire potential search region. The second step is to identify the potential initial areas for the target, as well as the respective probabilities for each area.

Defining the potential search region is a fairly simple process. Because the program uses a grid array, the shape of the region will always be a rectangle. In order to define the bounds of this rectangle, the user provides the latitude and longitude of the upper left corner of the region and the lower right corner of the region. The map-based planning system provides these coordinates to COINSS, along with the length and width of the region in terms of graphical pixels. COINSS then determines the search grid array size. Currently in COINSS, each grid square is 16 pixels by 16 pixels on the graphical display. The search grid array size is then computed by dividing the search region into the required number of 16x16 pixel squares to cover the region.

Defining the initial area for the target is a little more involved. A vector of geographic positions, generally denoted by longitude and latitude, is required. These positions will define the vertices for a polygon. This polygon denotes the initial area. This polygon does not need to be a regular polygon, and can be non-convex.

Each polygon is assigned a numerical percentage value, which is the probability that the target started in that area. The model takes this probability and distributes it equally among the grid squares located within the area. A grid square is considered to be located within the area if the center of the square is located within the area, otherwise it is considered to be located outside the area.

Multiple initial areas may exist, and these areas may overlap. Each initial area is created in the same manner. It is conceivable that the sum of the probabilities for all the initial areas would be less than one. This would occur if there exists a probability that the target could have initially started outside of the search region.

C. UPDATING THE PROBABILITY MAP

Two events require the update of the probability map. The first is the passage of time, which requires an update due to the movement of the target. The second is a report from a search team, which requires an update based upon the detection/non-detection of the target.

1. Time-Based Update

Time based updates adjust the probability distribution on the map due to the movement of the target. These updates can occur over any time interval. Smaller time intervals, such as one second, come closer to simulating a continuous time movement model. However, larger time steps reduce the computations, and over a large region with several hundred movement vectors, the loss of resolution is not significant.

COINSS uses a constant time step, currently set to fifteen minutes. COINSS only computes the time-based update when prompted by the user. This allows the user to control the pace of the model. The time-based update consists of three major steps:

1. Moving the velocity array probability elements from their current square to a new square, based upon their velocity and the time step increment.

2. Adjusting for changes in velocity after each probability element movement.

3. Correcting the velocity array probability elements for elevation.

The first step of the time-based update is to move the velocity array probability elements. This is done by sequentially accessing the velocity array of each square, $V_{x,y,i,j}$. Each element of the velocity array is moved the direction and distance that is dictated by the velocity vector and the corresponding time step increment, T_1, to a new location, $S_{xnew,ynew}$.

$$xnew = x + i * T_1$$

$$ynew = y + j * T_1$$

The probability at this new location is evenly distributed over a square region, centered on the new geographical position (Figure 3-1). The probability is then assigned to each grid square in proportion to the area of the probability square that is located within the grid square [6].

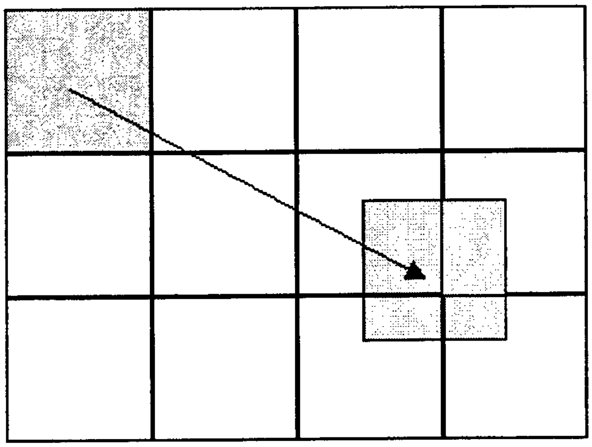


Figure 3-1. Movement and distribution of velocity array probability element

The next step is to adjust the current velocity of the probability element. The velocity of the target is modeled to change according to a Markov chain. To model the velocity changes as a Markov chain, each velocity element is recognized as a state. The probability of changing velocity from state *i* to state *inew*, which can be read as changing from velocity *i* to velocity *inew*, is represented by a Markov transition matrix, $MM_{i,inew}$ [13]. The transition probabilities of this matrix are based on the target type of the search. Appendix A discusses the five target types defined in COINSS and provides the Markov transition matrix for each type. Figure 3-2 is the transition matrix for target type 1, which is a child age 1-6.

Transition matrix (feet per second velocity)

	-4	-3	-2	-1	0	1	2	3	4
-4	0.01	0.06	0.43	0.43	0.06	0.01	0	0	0
-3	0	0.01	0.11	0.76	0.11	0.01	0	0	0
-2	0	0.01	0.07	0.6	0.27	0.04	0.01	0	0
-1	0	0.01	0.04	0.27	0.6	0.07	0.01	0	0
0	0	0	0.01	0.11	0.76	0.11	0.01	0	0
1	0	0	0.01	0.07	0.6	0.27	0.04	0.01	0
2	0	0	0.01	0.04	0.27	0.6	0.07	0.01	0
3	0	0	0	0.01	0.11	0.76	0.11	0.01	0
4	0	0	0	0.01	0.06	0.43	0.43	0.06	0.01

Figure 3-2. Transition Matrix for a child age 1-6.
Each row of this matrix sums to 1.

This matrix is applied individually to both the x and y components of the velocity. For example, from Figure 3-2, the probability of transitioning from velocity -2 (ft/sec) to -1 (ft/sec) is .6, and is written as $MM_{-2,-1}$. The probability element, labeled $V_{x,y,i,,j}$, can then be distributed to the velocity array of its new square, $V_{xnew,ynew,inew,jnew}$, according to the following:

$$V_{xnew,ynew,inew,jnew} = \sum_{\substack{x'=xnew-T_1*i,\\ y'=ynew-T_1*j,\\ i,j}} V_{x',y',i,j} * MM_{i,inew} * MM_{j,jnew}$$

For example, suppose $V_{x,y,-2,-2}$ =.25. Then suppose that by following the example of Figure 1, it is computed that 40% of the moved square lies within $S_{xnew,ynew}$, or $V_{x',y',i,j}$

= .1, where x'=xnew-T_1*i and y'=ynew-T_1*j. By using the transition matrix for target type 1 from Figure 3-2, we obtain the resulting distribution shown in Figure 3-3.

						jnew				
		-4	-3	-2	-1	0	1	2	3	4
	-4	0	0	0	0	0	0	0	0	0
	-3	0	0.0001	0.0007	0.006	0.0027	0.0004	0.0001	0	0
	-2	0	0.0007	0.0049	0.042	0.0189	0.0028	0.0007	0	0
inew	-1	0	0.006	0.042	0.36	0.162	0.024	0.006	0	0
	0	0	0.0027	0.0189	0.162	0.0729	0.0108	0.0027	0	0
	1	0	0.0004	0.0028	0.024	0.0108	0.0016	0.0004	0	0
	2	0	0.0001	0.0007	0.006	0.0027	0.0004	0.0001	0	0
	3	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0

Figure 3-3. Resulting distribution of $V_{xnew,ynew,inew,jnew}$ when 40% of $V_{x,y,-2,-2}$ =.25 is moved to square $S_{xnew,ynew}$, for target type 1, small child age 1-6. The sum of all elements in this matrix is .1.

Once the probabilities have been moved and updated for all the squares in the search region, the final step of the time-based update adjusts the velocity array probability elements to account for elevation. For COINSS, elevation can be loaded from a CD-ROM that contains Digital Terrain Elevation Data Level 1 (DTED1) from the National Imagery and Mapping Agency. This data provides the elevation in meters and is available to the U.S. Government only. The coverage for this data is one measurement every three arc seconds, or approximately every 300 feet. If elevation data is not available, then the elevation is considered to be 0 meters for all locations.

The adjustments for elevation are done by again cycling through all the squares of the search region. At each square, the total probability of that square is computed.

$$Pinit_{x,y} = \sum_{i,j} V_{x,y,i,j}$$

The next step is to cycle through the velocity array **V** of that square. At each probability element, $V_{x,y,i,j}$, of the array, a slope is computed. This slope is computed in the direction that the element would move during the next time step (i.e., the slope from $S_{x,y}$ to $S_{xnew,ynew}$, where xnew = x +i*T_1 and ynew = y +j*T_1). If the slope is too steep for the particular target type to easily navigate in that direction, then the probability element, $V_{x,y,i,j}$, is reduced. If the probability is reduced, it is either reduced to 50% of its previous value or to zero, again depending on the severity of the slope and the target type (Figure 3-4.). These values were determined from judgements of human tendencies.

Target Type	50%	100%
1		.3
2	.3	.45
3	.3	.45
4	.45	.7
5	.45	.7

Figure 3-4. Slopes at which the probability of moving is reduced either 50% or 100%

After all elements of the array **V** in Square $S_{x,y}$ have been updated, the total probability of the square must be readjusted back to its original value, $Pinit_{x,y,}$ by computing the new total square probability, and re-scaling each element of the velocity array. Let $V'_{x,y,i,j}$ be the velocity array elements after adjustment for elevation, and let

$$Pnew_{x,y} = \sum_{i,j} V'_{x,y,i,j}$$

then

$$V_{x,y,i,j} = \frac{Pinit_{x,y} * V'_{x,y,i,j}}{Pnew_{x,y}}; \forall i, j$$

2. Search-Based Update

Each search conducted in the search region is applied to the model independent of any other search. The variables required for a search are the search unit name, the search area, and the probability of detection. The search unit name is a label to help identify this particular search instance. The search area is a vector of geographic positions, generally denoted by longitude and latitude. These positions will define the vertices for a polygon. Unlike the search region, the search area is not required to be a rectangle and it may be a non-convex polygon. The probability of detection is a value between 0 and 1 that is provided by the user.

Search based updates adjust the probability distribution for all the squares in the logical map due to search unit results. These updates use Bayes theorem to account for unsuccessful search results. This result is computed by using conditional probability rules and Bayes theorem. The following two events are defined:

$D_{x,y}$: The target is detected at location $S_{x,y}$ by the search effort allocated to $S_{x,y}$
$T_{x,y}$: The target's current location is $S_{x,y}$

Then the probability of detection for the search unit, conditioned on the event that the target is located in the searched area, is defined as:

$$P[D_{x,y} \mid T_{x,y}]$$

and the probability of non-detection, conditioned on the event that the target is located in the searched area is defined as:

$$P[D'_{x,y} \mid T_{x,y}] = 1 - P[D_{x,y} \mid T_{x,y}]$$

where the superscript denotes negation. Likewise:

$$P[D_{x,y} \big| T_{x',y'}] = 0, \forall (x, y) \neq (x', y')$$

$S_{x,y}$ is determined to be located within the search area if and only if the center of square $S_{x,y}$ is located within the search area polygon.

When a search party reports that they have searched an area and did not find the target, the model is updated. Each square in the search region must first be updated to show the probability that the target is located in that region and was not detected by the search party. This is defined as:

$$P[T_{x',y'} \cap D'_{x,y}] = P[D'_{x,y} | T_{x',y'}]P[T_{x',y'}]$$

Using Bayes theorem, we arrive at a posterior probability that is the probability of the target being located at location $S_{x',y'}$, given that it was not detected at location $S_{x,y}$. This is defined as:

$$P[T_{x',y'} \mid D'_{x,y}] = \frac{P[T_{x',y'}]P[D'_{x,y} \mid T_{x',y'}]}{\sum_{x'',y''} P[T_{x'',y''} \cap D'_{x'',y''}]}, \forall x, y, x', y'$$

It is worth noting that all squares in the search region are being updated, not just the squares located within the search area. For all the locations $S_{x',y'}$ located outside the search area, $\mathrm{P}[\mathrm{D}'_{x,y} | \mathrm{T}_{x',y'}] = 1$, and therefore $\mathrm{P}[\mathrm{T}_{x',y'} \cap \mathrm{D}'_{x,y}] = \mathrm{P}[\mathrm{T}_{x',y'}]$. These values are used in the above equation to create a new probability distribution for the location of the target.

IV. VISUAL REPRESENTATION OF THE MODEL

A. COMPONENT-BASED DESIGN

COINSS was designed to be a component-based system. COINSS provides the algorithms, methods, and logic for executing a SAR mission plan. It is designed to take advantage of well-defined reusable components from existing map-based planning systems, such as the ability to read latitude and longitude or to provide overlay displays.

The "Loosely Coupled Components" project designed by Dr. G.H. Bradley and Dr. A.H. Buss, both of the Naval Postgraduate School, provides one such map-based planning system [4]. Their project creates planning systems by integrating various components as required. Components can be maps, algorithms, data overlays, or any other item necessary to complete a task. The nature of the system is such that the planner only needs to assemble the distinct components needed at that time. If a need for a new component arises during a planning operation, the planner can obtain or implement that component and add it to the planning system without having to restart the entire operation. It also allows designers to concentrate on creating algorithms or methods to solve problems, and not continually "re-invent the wheel" by developing a graphical interface for their specific problem.

B. BASIC CONSTRUCTION OF THE VIEW

1. Grid Dimensions

To present a COINSS probability map on a visual display, a translation from the squares on the logical probability map to the pixels on the monitor is required. A standard method of presenting probability maps on a visual display is by using different colors or shades of a single color to represent the probabilities. The visual display square size needs to be large enough to allow visual discrimination of color and the squares must be small enough to provide useful search information (i.e., a search region with only two squares would not provide much meaningful search information). Finally, the logical array needs to be small enough to provide movement and search updates in a reasonable amount of computation time. COINSS was developed on a PC with a 166 MHZ CPU. A 16x16 pixel square is a reasonable choice for this computer.

2. Probability Display

COINSS uses 10 shades of the color red to discriminate among the different quantitative probability values. The red shades are equally spaced, which is achieved by manipulating the RGB color value of the pixels. The RGB value ranges from 0 to 255. The shades were created by modifying the values in increments of 25 from 5 to 255. Red was chosen as the color to use, and the darkest red was achieved with RGB values 255|0|0, while the lightest shade was achieved with RGB values 255|225|225 [8].

The variable maxSquareProb contains the value of the largest probability in the search region.

$$\text{maxSquareProb} = \max\{\sum_{i,j} V_{x,y,i,j}\}, \forall x, y$$

Each square is assigned a color, which is based on the probability of the square compared to the maxSquareProb. The maxSquareProb is assigned the purest color value (255|0|0). Squares containing a probability that is 90%-99% of maxSquareProb receive the next highest purest value (255|25|25). This continues at 10% intervals down to the squares that have probabilities that are between 10%-19% of the maxSquareProb. These squares receive the least pure color value (255|225|225). The squares with probability less than 10% of the maxSquareProb are assigned to be completely transparent, and appear as blank squares on the probability map.

COINSS also makes use of opaqueness. The opaqueness of a cell determines how well you can see through the cell on the top layer and view images on the layer(s) below it. An opaque value of 255 is completely opaque and does not allow the viewer to see any part of the layer below it, while an opaque value of 0 is completely transparent [8]. COINSS currently assigns each colored square in the probability map an opaqueness value of 190. This allows the layer immediately below the probability map to be seen, with little loss in color discrimination.

C. INTERACTING WITH THE VIEW

1. Selecting a Geographic Map

COINSS is designed to operate in conjunction with a map-based planning system, such as the Loosely Coupled Components system created by Bradley and Buss [4]. The map-based system is responsible for obtaining a geographic map from any available source. The planning system is also responsible for providing an interface to COINSS, specifically, a reference to the latitude and longitude of the locations on the map, as well as a representation of the scale of the map provided in pixels per unit longitude. The map can have any geographic or topological features or symbols. This map is merely a visual reference for the planner, and will have no impact to the logical computations that take place in the model.

2. Defining the Search Region

Identifying the bounds for the search region is the next step. This is a user-defined region that is provided by the map-based planning system. The search region is required to be a rectangular region, and is defined by two points, the upper left corner and the lower right corner. A simple method is to allow a mouse click at the location for the upper left corner, followed by another click at the location for the lower right corner. These two values are then provided to COINSS to define the search region.

3. Defining the Initial Areas

Initial areas are also user-defined areas that are provided by the map-based planning system. Again the mouse is used to outline the initial areas. As discussed earlier, a vector of geographic positions is required. These positions define the vertices for a polygon. The mouse clicks define each vertex, in order. Once the polygon is completed, it is assigned a numerical value, which is the probability that the target started in that area. COINSS takes this probability and distributes it equally among the grid squares located within the region, and across all the possible velocity vectors.

This is an iterative process, conducted on each initial area in a sequential manner. Each subsequent initial area probability is added to the existing probability map. If two or more initial areas overlap, then the probability associated with the overlapped area is equal to the sum of the individual probabilities for that square.

4. Defining the Search Area

Search areas are the third case of user-defined areas. These are also provided by the map-based system, and are created in the same manner as the initial areas. The numerical value assigned to each search area is the probability that the unit searching that area will locate the target if the target is in that area. This probability is used to apply Baysian updates to the probability map.

V. CONCLUSIONS AND RECOMMENDATIONS

COINSS provides an inland search and rescue model which implements target movement during the search execution. Target movement is modeled as a discrete time Markov chain, based upon the type of the target. Geographic elevation is also accounted for in the movement model by limiting movement in areas with large changes in elevation.

COINSS is a component-based system, designed for implementation with a map-based planning system. This design demonstrates that a component-based design can reuse components. Allowing different component systems to use the same map-based system interface increases the efficiency of the hardware, reduces user training, and reduces software cost and redundancy.

This thesis provides several areas for future research. The movement model is very rudimentary. The COINSS Markov movement model is based upon historical data that only provides the total distance traveled from initial datum [9][17]. More detailed study of target movement rates and tendencies would improve the movement model.

Integrating additional information regarding the terrain would also improve the movement model. Identification of impediments such as heavy brush, rivers, and lakes, or easier traveling terrain such as trails, paths, and roads would improve the realistic modeling of movement, and create a more realistic probability model.

APPENDIX A. MARKOV MOVEMENT MATRICES

At each movement phase update, the velocity vector of the target changes as discussed in Chapter III. To implement the movement model using a Markov chain, a discretized version of a Kalman filter was created [22].

Using a discrete version of the Ornstein-Uhlenbeck (OU) process [22], velocity is modeled as

$$X_t = cX_t + W_t$$

where X_t is the velocity of the target at time t, W_t is an independent identically distributed normal random variable with mean 0 and variance q, and c is a constant. The values for c and q determine the variance of the velocity for the target. Furthermore,

$$c = e^{-\frac{\Delta}{\tau}}$$

where Δ is the time interval at which updates are made, and 1/τ is the expected rate (in time units) that the target changes velocity [22]. Δ is an artificial variable which is determined by COINSS, while τ is a characteristic of the target that is estimated from the data.

The variance of the total movement of the target can be expressed as [19]

$$Var(X(\delta)) = \frac{1}{2}(2s^2\tau)[2\delta - \tau[3 - 4e^{-\frac{\delta}{\tau}} + e^{-2\frac{\delta}{\tau}}]]$$

where s^2 is a target characteristic which identifies the variance of velocity, and δ is the time interval over which the target has moved. By computing the variance for different target types from Syrotuck [17] and Hill [9] (Figure A-1.) and estimating values for δ and τ, s^2 can be solved for, and then q can be computed from [22]:

$$q = s^2(1-c^2)$$

Target Type	Hill Var	Syrotuck Var
1	1.23	6.34
2	6.34	8.78
3	2.19	N/A
4	30.05	10.62
5	21.09	22.47

Figure A-1. Variance of total movement (in miles) for found targets in Hill[9] and Syrotuck[17].

The data provided by Hill and Syrotuck in Figure A-1 have two shortcomings. First, the data is only for found targets. As such, we must assume that all targets move in a fashion similar to that of the found targets. Second, the data has no time associated with it. The longer a target remains undetected, the larger the variance of his movement. However, this is the best data available on lost targets. Therefore, the data was used, along with a generic view of human movement patterns to create one model of movement.

After c and q have been estimated, the Markov transition matrix can be constructed using the assumption that velocity changes will vary according to a standard normal distribution. Figure A-2 provides a chart of all the values for τ, which were estimated based upon human tendencies, and s^2, which was computed with an estimated δ of 24 hours. Figures A-3 through A-7 are the transition matrices for each target type when $\Delta = 15$ minutes. The indices for the transition matrices are in feet per second. The target types are [9]:

1: children under age 6,

2: children age 7-12,

3: despondents/walkaways,

4: hunters,

5: hikers/adults.

Target Type	τ(sec)	S^2 (ft^2/sec^2)
1	900	1.155
2	1800	.8122
3	1800	.8122
4	3600	.5078
5	3600	1.074

Figure A-2. Computed values for target movement

	-4	-3	-2	-1	0	1	2	3	4
-4	0.02	0.14	0.34	0.34	0.14	0.02	0	0	0
-3	0	0.04	0.24	0.44	0.24	0.04	0	0	0
-2	0	0.03	0.19	0.39	0.29	0.09	0.01	0	0
-1	0	0.01	0.11	0.31	0.37	0.17	0.03	0	0
0	0	0	0.04	0.24	0.44	0.24	0.04	0	0
1	0	0	0.03	0.17	0.37	0.31	0.11	0.01	0
2	0	0	0.01	0.09	0.29	0.39	0.19	0.03	0
3	0	0	0	0.04	0.24	0.44	0.24	0.04	0
4	0	0	0	0.02	0.14	0.34	0.34	0.14	0.02

Figure A-3. Transition matrix for target type 1

	-4	-3	-2	-1	0	1	2	3	4
-4	0.13	0.37	0.37	0.11	0.02	0	0	0	0
-3	0.02	0.17	0.46	0.28	0.07	0	0	0	0
-2	0	0.07	0.28	0.46	0.17	0.02	0	0	0
-1	0	0.02	0.18	0.36	0.38	0.06	0	0	0
0	0	0	0.02	0.23	0.51	0.23	0.02	0	0
1	0	0	0	0.06	0.38	0.36	0.18	0.02	0
2	0	0	0	0.02	0.17	0.46	0.28	0.07	0
3	0	0	0	0	0.07	0.28	0.46	0.17	0.02
4	0	0	0	0	0.02	0.11	0.37	0.37	0.13

Figure A-4. Transition matrix for target type 2

	-4	-3	-2	-1	0	1	2	3	4
-4	0.13	0.37	0.37	0.11	0.02	0	0	0	0
-3	0.02	0.17	0.46	0.28	0.07	0	0	0	0
-2	0	0.07	0.28	0.46	0.17	0.02	0	0	0
-1	0	0.02	0.18	0.36	0.38	0.06	0	0	0
0	0	0	0.02	0.23	0.51	0.23	0.02	0	0
1	0	0	0	0.06	0.38	0.36	0.18	0.02	0
2	0	0	0	0.02	0.17	0.46	0.28	0.07	0
3	0	0	0	0	0.07	0.28	0.46	0.17	0.02
4	0	0	0	0	0.02	0.11	0.37	0.37	0.13

Figure A-5. Transition matrix for target type 3

	-4	-3	-2	-1	0	1	2	3	4
-4	0.2	0.66	0.14	0	0	0	0	0	0
-3	0.04	0.33	0.53	0.1	0	0	0	0	0
-2	0	0.07	0.43	0.43	0.07	0	0	0	0
-1	0	0	0.14	0.48	0.28	0.1	0	0	0
0	0	0	0.02	0.23	0.51	0.23	0.02	0	0
1	0	0	0	0.1	0.28	0.48	0.14	0	0
2	0	0	0	0	0.07	0.43	0.43	0.07	0
3	0	0	0	0	0	0.1	0.53	0.33	0.04
4	0	0	0	0	0	0	0.14	0.66	0.2

Figure A-6. Transition matrix for target type 4

	-4	-3	-2	-1	0	1	2	3	4
-4	0.27	0.46	0.24	0.03	0	0	0	0	0
-3	0.11	0.31	0.39	0.17	0.02	0	0	0	0
-2	0.02	0.13	0.35	0.35	0.13	0.02	0	0	0
-1	0	0.03	0.18	0.4	0.3	0.08	0.01	0	0
0	0	0	0.03	0.24	0.46	0.24	0.03	0	0
1	0	0	0.01	0.08	0.3	0.4	0.18	0.03	0
2	0	0	0	0.02	0.13	0.35	0.35	0.13	0.02
3	0	0	0	0	0.02	0.17	0.39	0.31	0.11
4	0	0	0	0	0	0.03	0.24	0.46	0.27

Figure A-7. Transition matrix for target type 5

APPENDIX B. COINSS SCENARIO

This is a walk through of COINSS implemented on a map-based planning system. The system used to implement COINSS in this scenario is Special Operations Forces/Loosely Coupled Components (SOFLCC) [2].

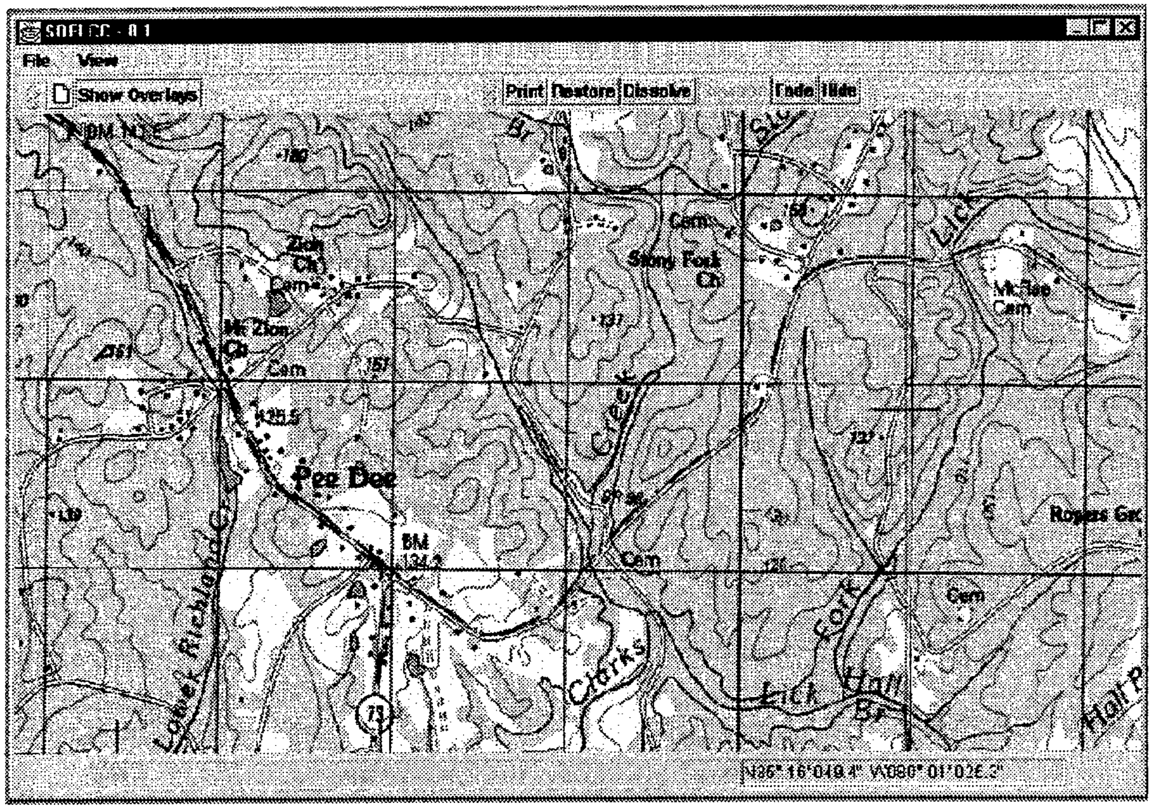


Figure B-1. The standard map-based planning system. The latitude and longitude of the mouse position is displayed in the lower right of the display.

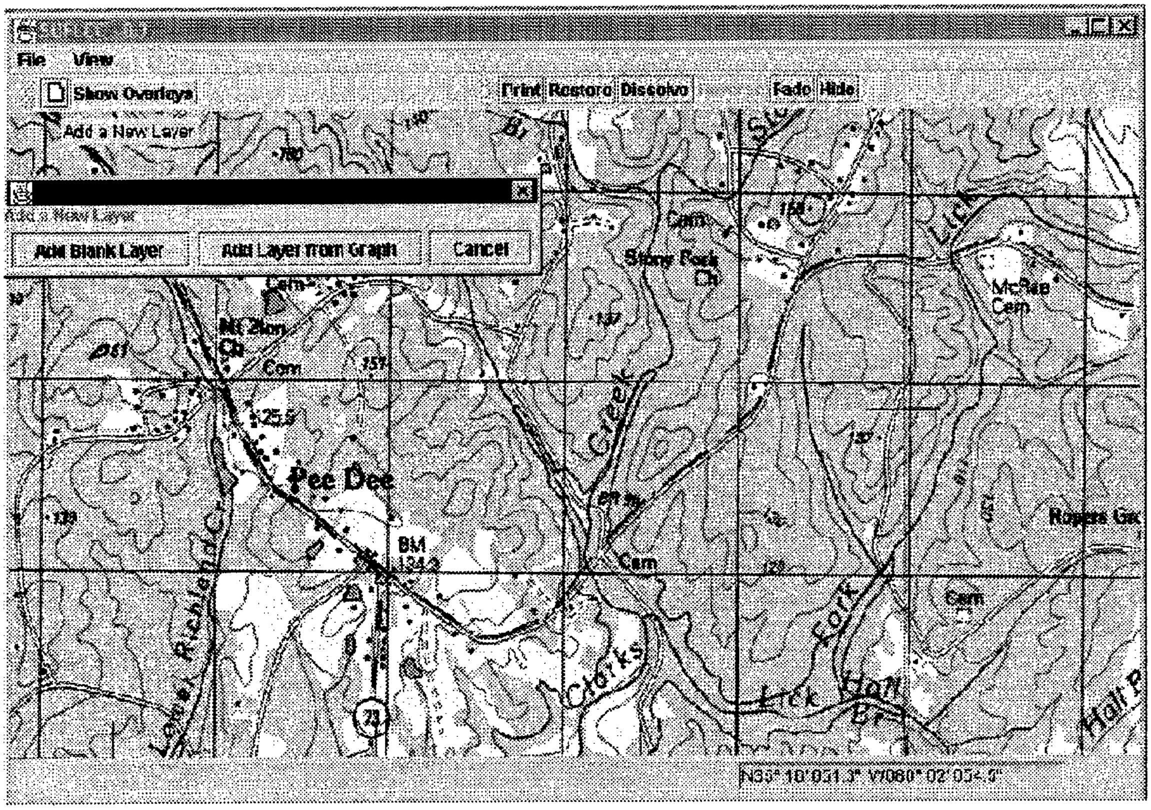


Figure B-2. To add a new overlay to the map, click the "Add a new Overlay" Icon. This then provides the option to add a (new) blank layer, or add a (pre-existing) layer from a file.

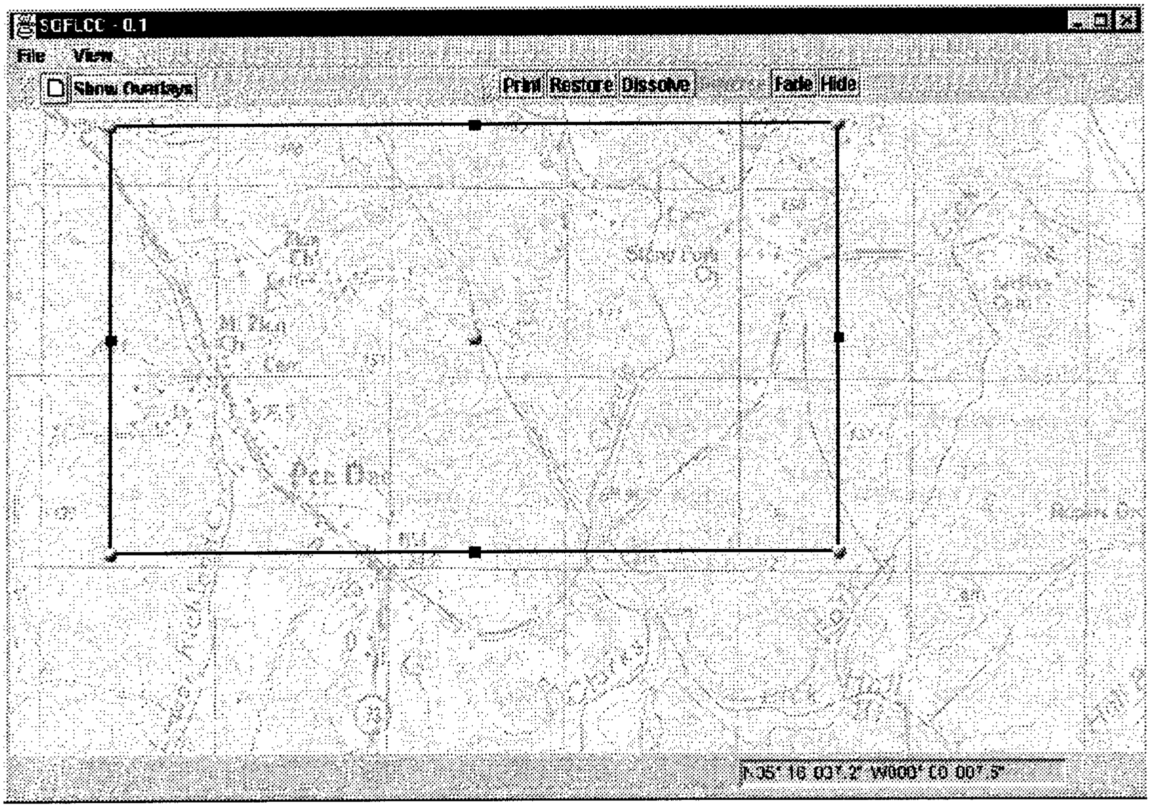
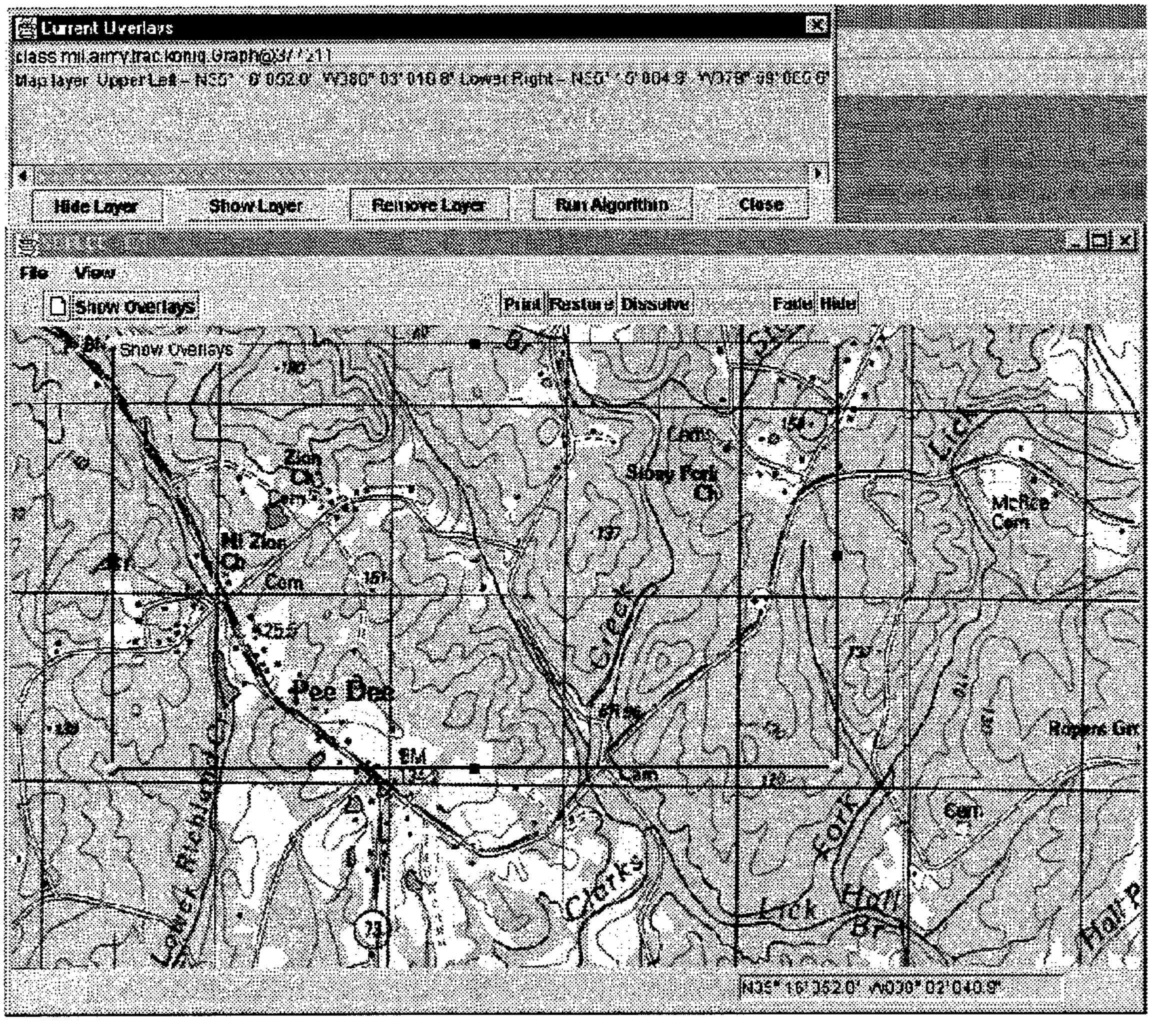


Figure B-3. The search region is created by clicking once on the map at the north-west corner of the search region, then double-clicking at the south-east corner of the region. The map has been faded to provide a clearer view of the search region.

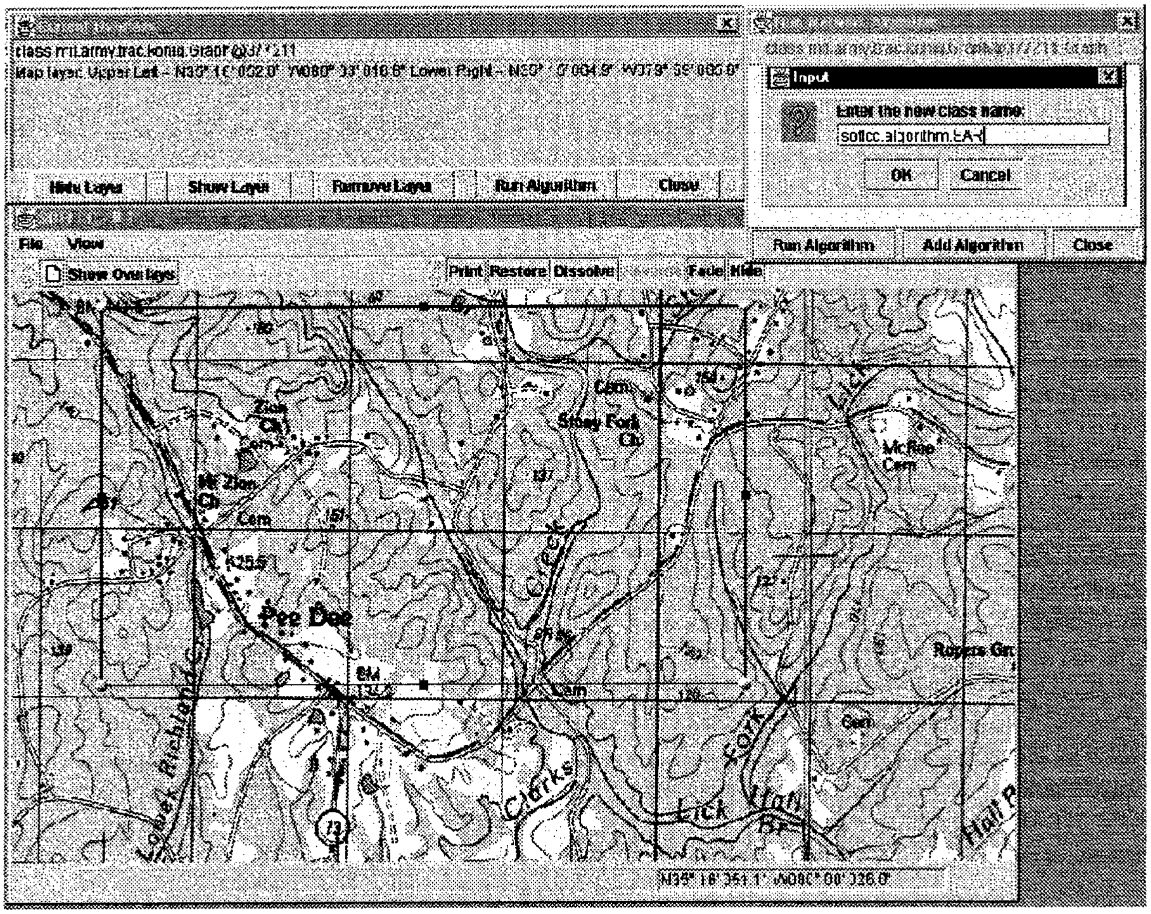


class mil.army.trac.konig.Graph@3/ '211

Map layer: UpperLeft – N35° 0' 052.0' W080° 03' 010.8' Lower Right – N35° 0' 004.9' W079° 59' 060.6'

N35° 16' 352.0' W080° 02' 040.9'

Figure B-4. Select the Show Overlay icon to select the search region for initialization. Select the Run Algorithm option to execute a SAR algorithm.



class mil.army.trac.konig.Grapr@3?*211

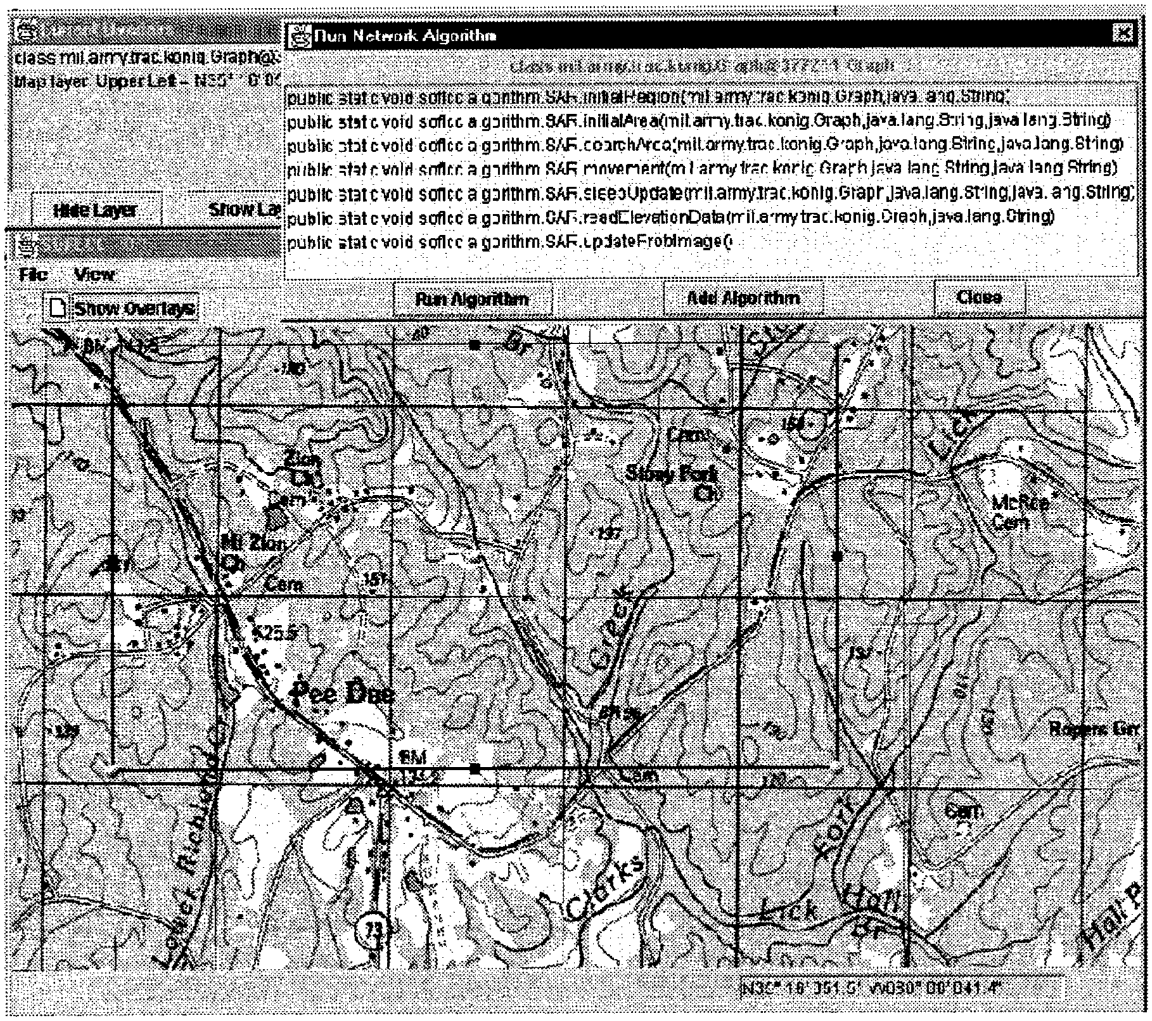
Map layer: Upper Left - N35° 1?' 052.0" W080° 33' 010.6" Lower Right - N35° ?' 004.9" W079° 55' 000.0"

Input

Enter the new class name:

soflcc.algorithm.EAR

Figure B-5. Select the Add Algorithm option to dynamically load the SAR algorithms.



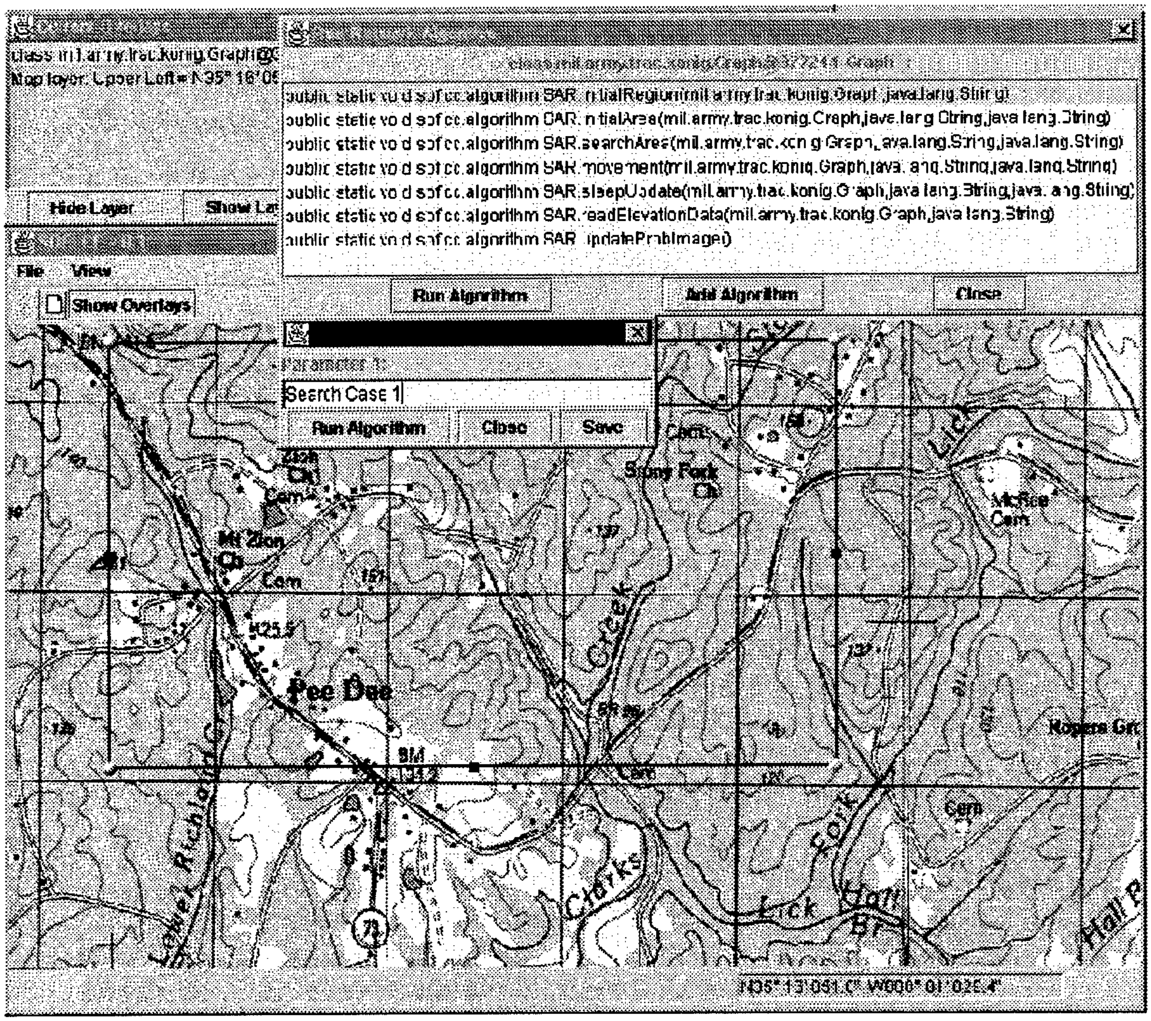
class mil.army.trac.konig.Graph@
Map layer: Upper Left – N35° 0' 0

class mil.army.trac.konig.Graph@3772[illegible] Graph

```
public static void softcc.algorithm.SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void softcc.algorithm.SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softcc.algorithm.SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softcc.algorithm.SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softcc.algorithm.SAR.sleepUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softcc.algorithm.SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void softcc.algorithm.SAR.updateProbImage()
```

N35° 16' 351.5' W080° 00' 041.4"

Figure B-6. Select the SAR.initialRegion to create the search region.



class mil.army.trac.konig.Graph@
Map layer: Upper Left = N35° 16'0

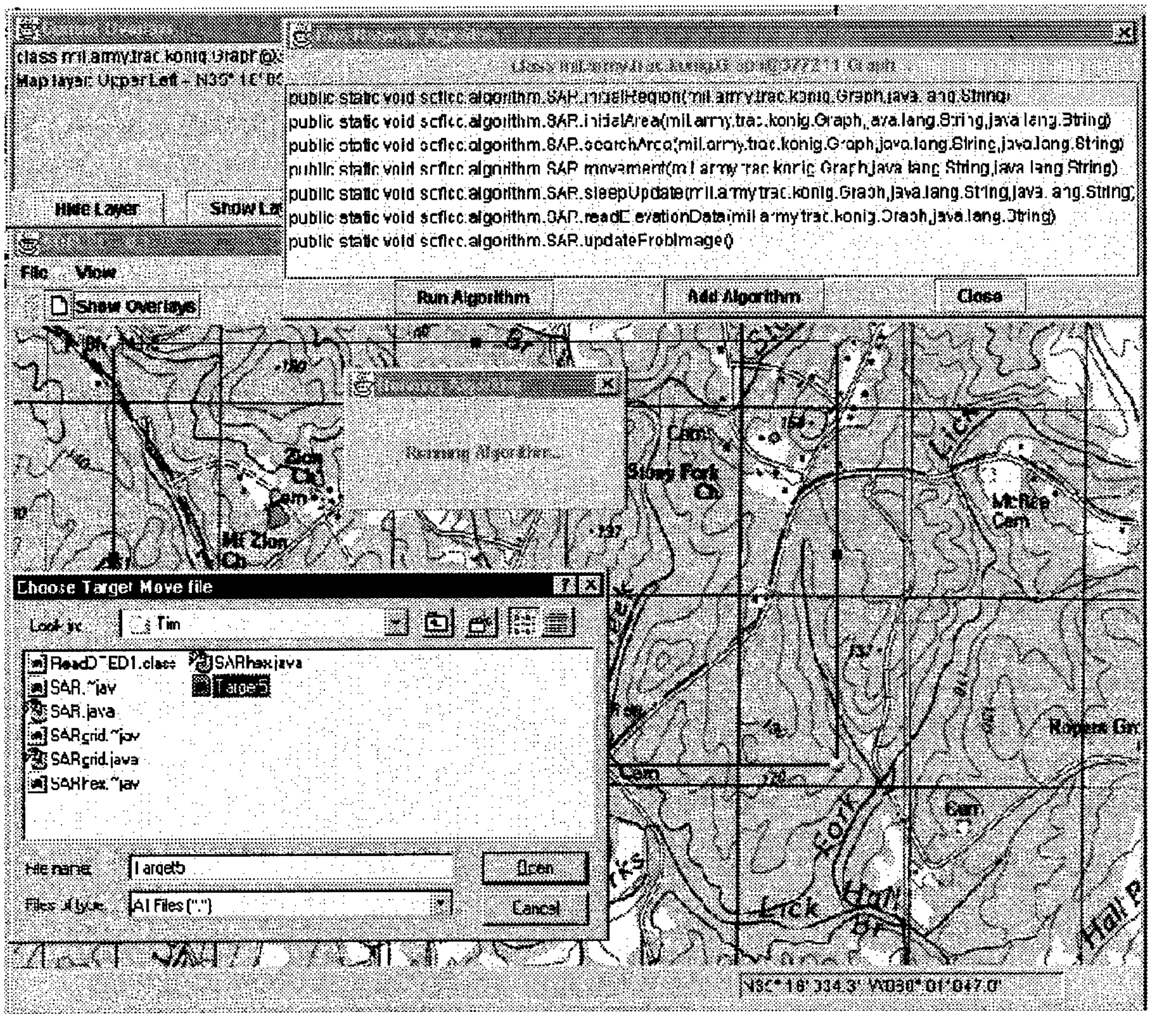
class mil.army.trac.konig.Graph@377241 Graph

```
public static void sofcc.algorithm SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void sofcc.algorithm SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm SAR.sleepUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void sofcc.algorithm SAR.updateProbImage()
```

Parameter 1:

Search Case 1

Figure B-7. The initialRegion algorithm requires a name as an input parameter.



```
class mil.army.trac.konig.Graph@
Map layer: Upper Left - N35° 18'0
```

```
public static void scficc.algorithm.SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void scficc.algorithm.SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void scficc.algorithm.SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void scficc.algorithm.SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void scficc.algorithm.SAR.sleepUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void scficc.algorithm.SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void scficc.algorithm.SAR.updateProbImage()
```

Choose Target Move file

Look in: Tim

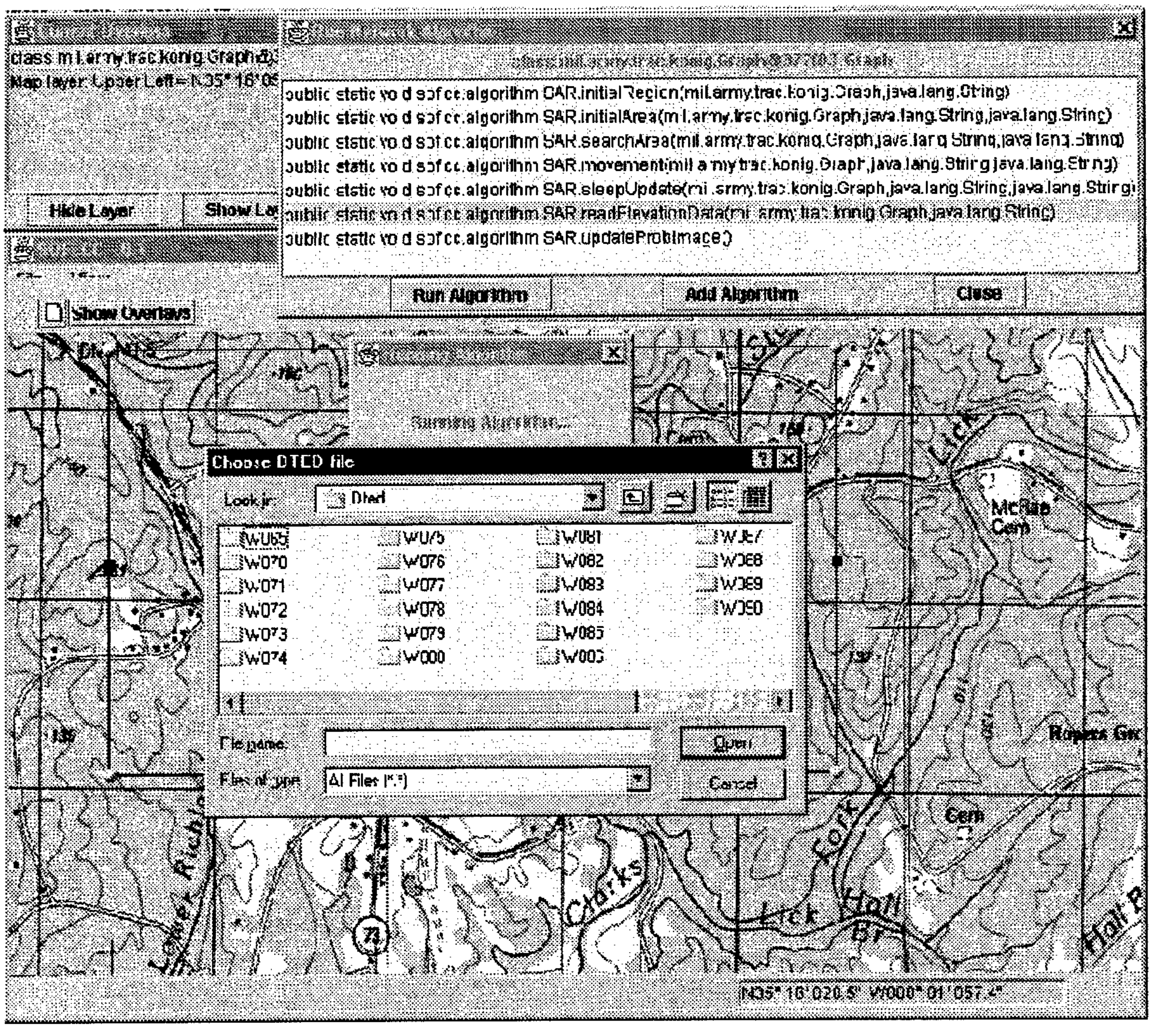
```
ReadDTED1.class    SARhex.java
SAR.~jav           Target5
SAR.java
SARgrid.~jav
SARgrid.java
SARhex.~jav
```

File name: Target5

Files of type: All Files (*.*)

N35° 18' 334.3' W080° 01' 047.0'

Figure B-8. The system queries for the type of search target, which is contained in a file.



```
public static void sofcc.algorithm.SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void sofcc.algorithm.SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm.SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm.SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm.SAR.sleepUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm.SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void sofcc.algorithm.SAR.updateProbImage()
```

Figure B-9. If elevation data is available, the user can import that data into the system.

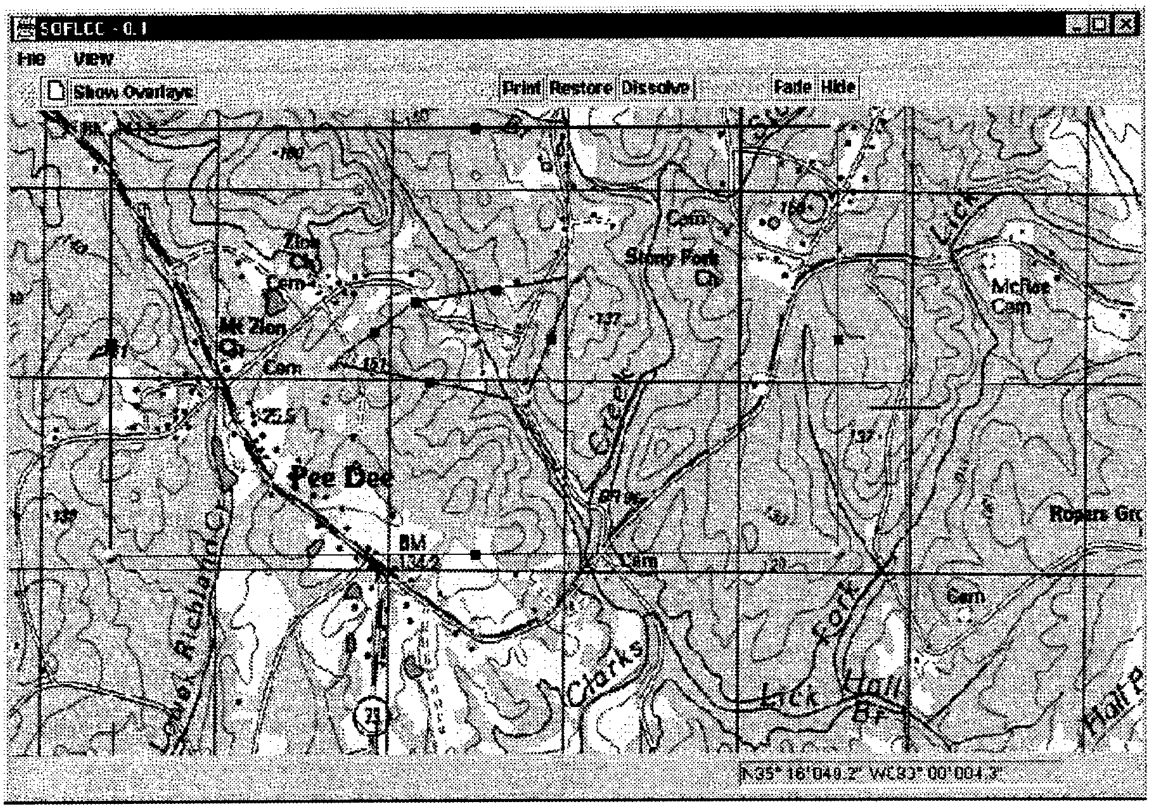
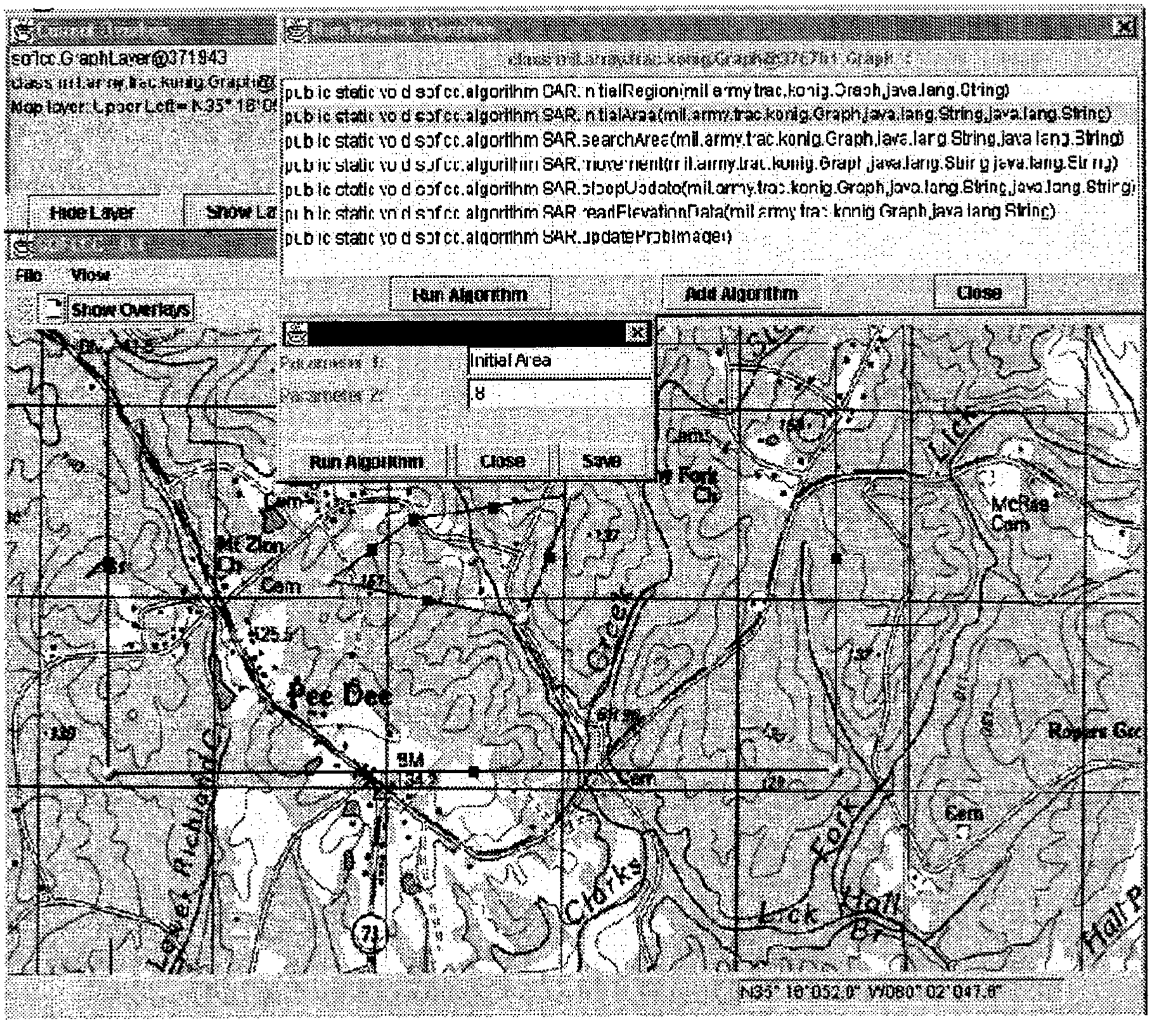


Figure B-10. Add a new overlay to create the initial area. The area is created by single clicks of the mouse at each location of the map which is a vertex for the area. Double click at the last vertex to complete the area.



sofcc.GraphLayer@371943

class mil.army.trac.konig.Graph@

Map layer: Upper Left= N35° 18'0

class mil.army.trac.konig.Graph@37...

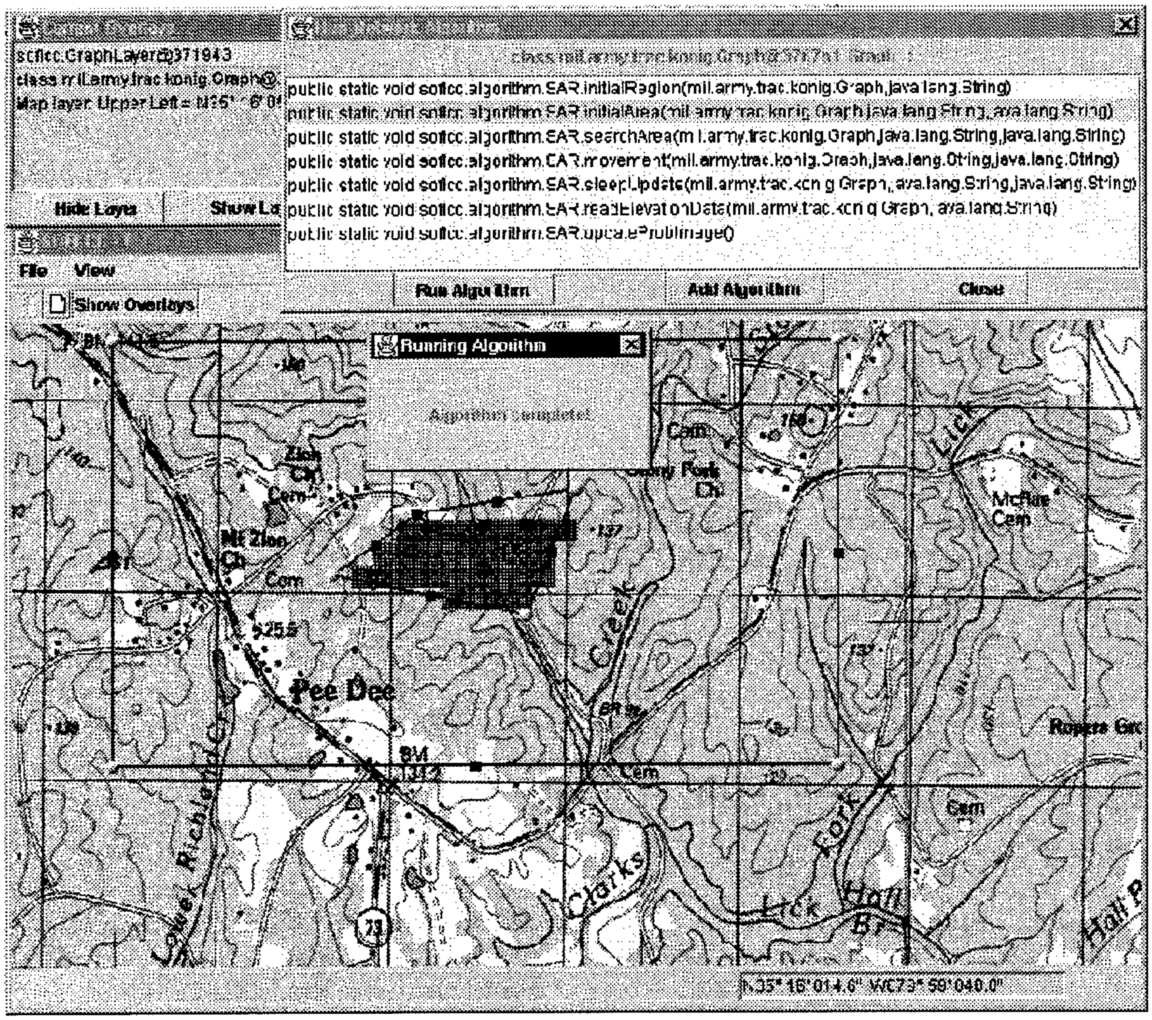
```
public static void sofcc.algorithm.SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void sofcc.algorithm.SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm.SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm.SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void sofcc.algorithm.SAR.slopeUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.Strin
public static void sofcc.algorithm.SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void sofcc.algorithm.SAR.updateProbImage()
```

Parameter 1: Initial Area

Parameter 2: .8

N35° 18'052.0" W080° 02'047.6"

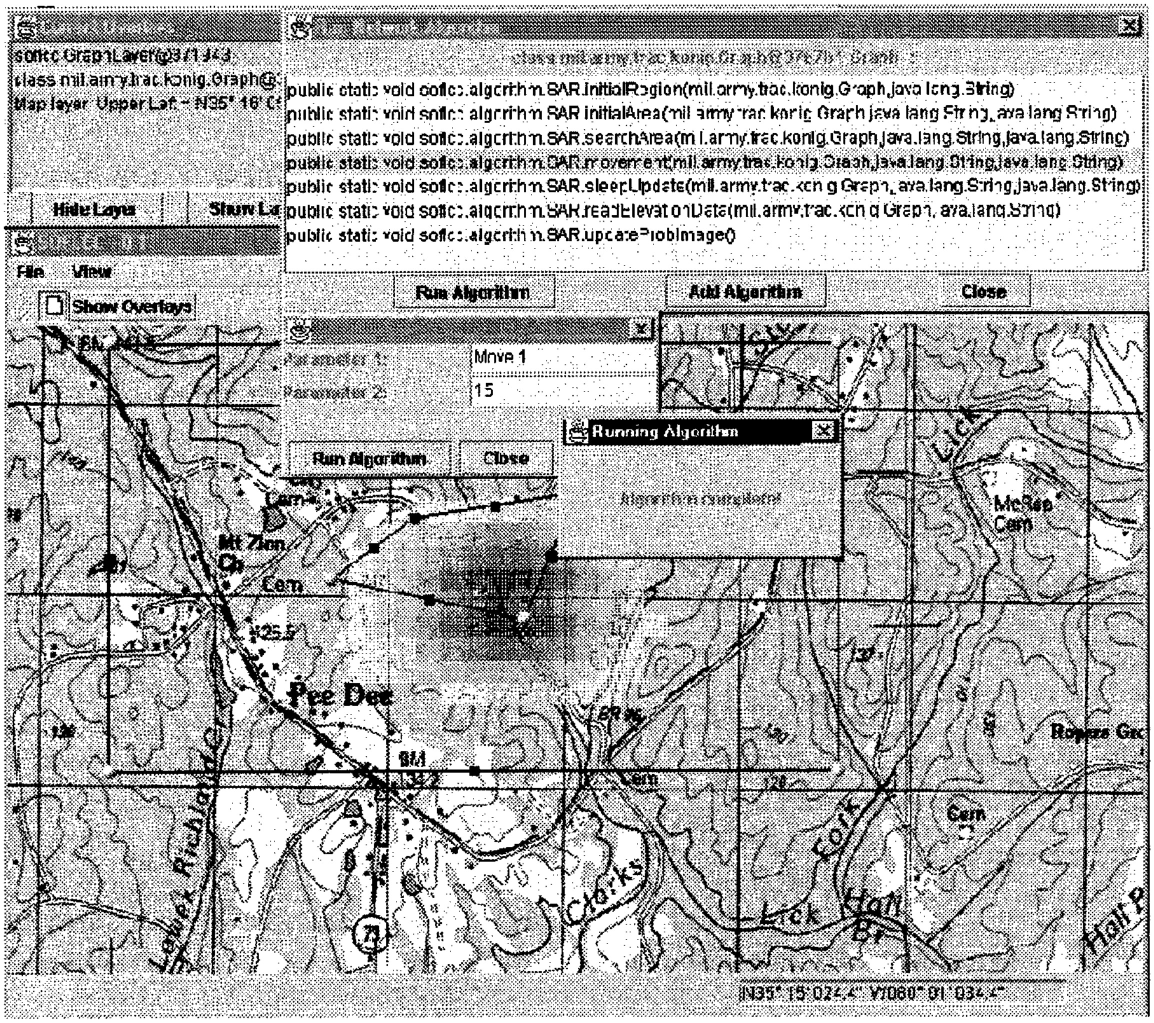
Figure B-11. Use the Show Overlay icon to select the initial area graph, and run the initialArea algorithm. This algorithm requires a name and a probability between 0 and 1 as parameters.



```
soficc.GraphLayer@371943
class mil.army.trac.konig.Graph@
Map layer: Upper Left = N35° - 6' 0
```

```
public static void soficc.algorithm.SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void soficc.algorithm.SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void soficc.algorithm.SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void soficc.algorithm.SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void soficc.algorithm.SAR.sleepUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void soficc.algorithm.SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void soficc.algorithm.SAR.updateProbImage()
```

Figure B-12. The probability map updates to show the current probability distribution.



softc.GraphLayer@371343
class mil.army.trac.konig.Graph@
Map layer: Upper Left = N35° 16' C

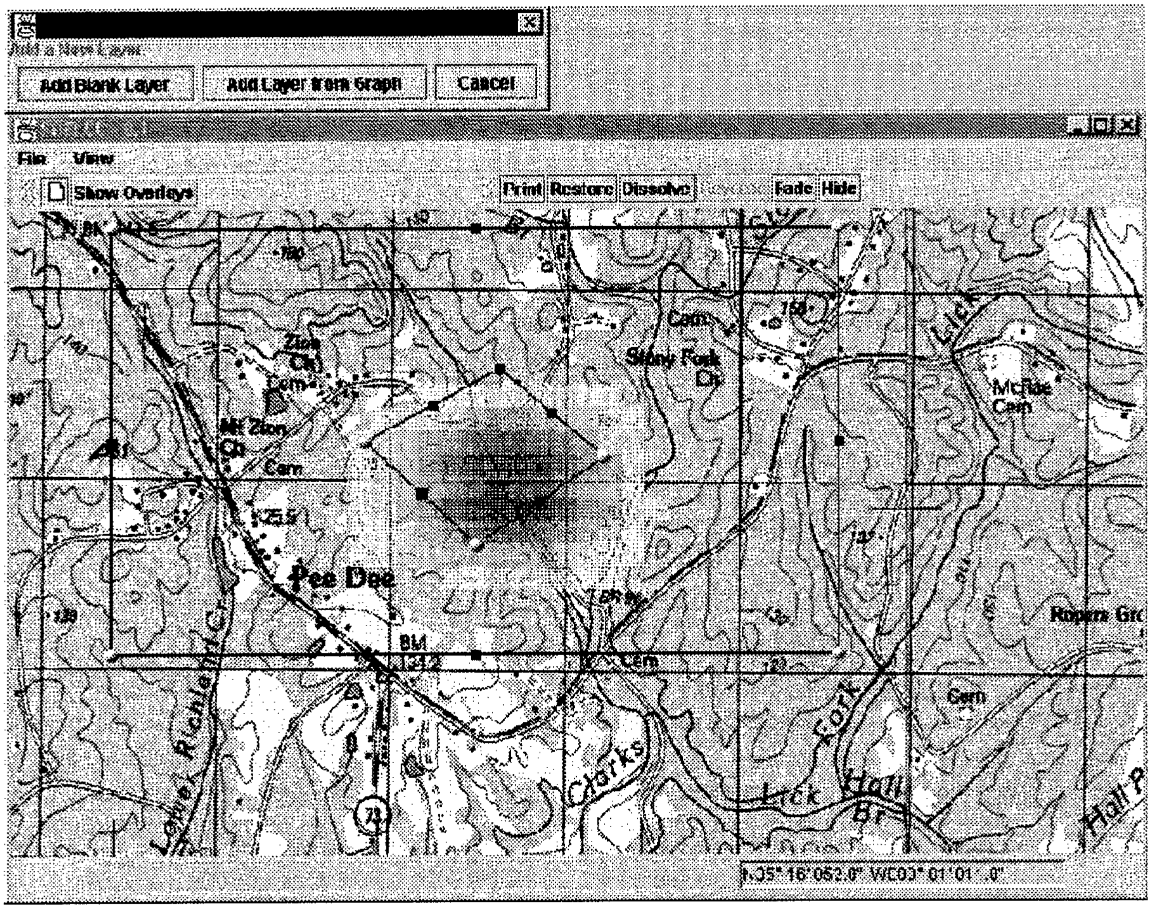
```
public static void softc.algorithm.SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void softc.algorithm.SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.sleepUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void softc.algorithm.SAR.updateProbImage()
```

Parameter 1: Move 1
Parameter 2: 15

Running Algorithm
Algorithm completed!

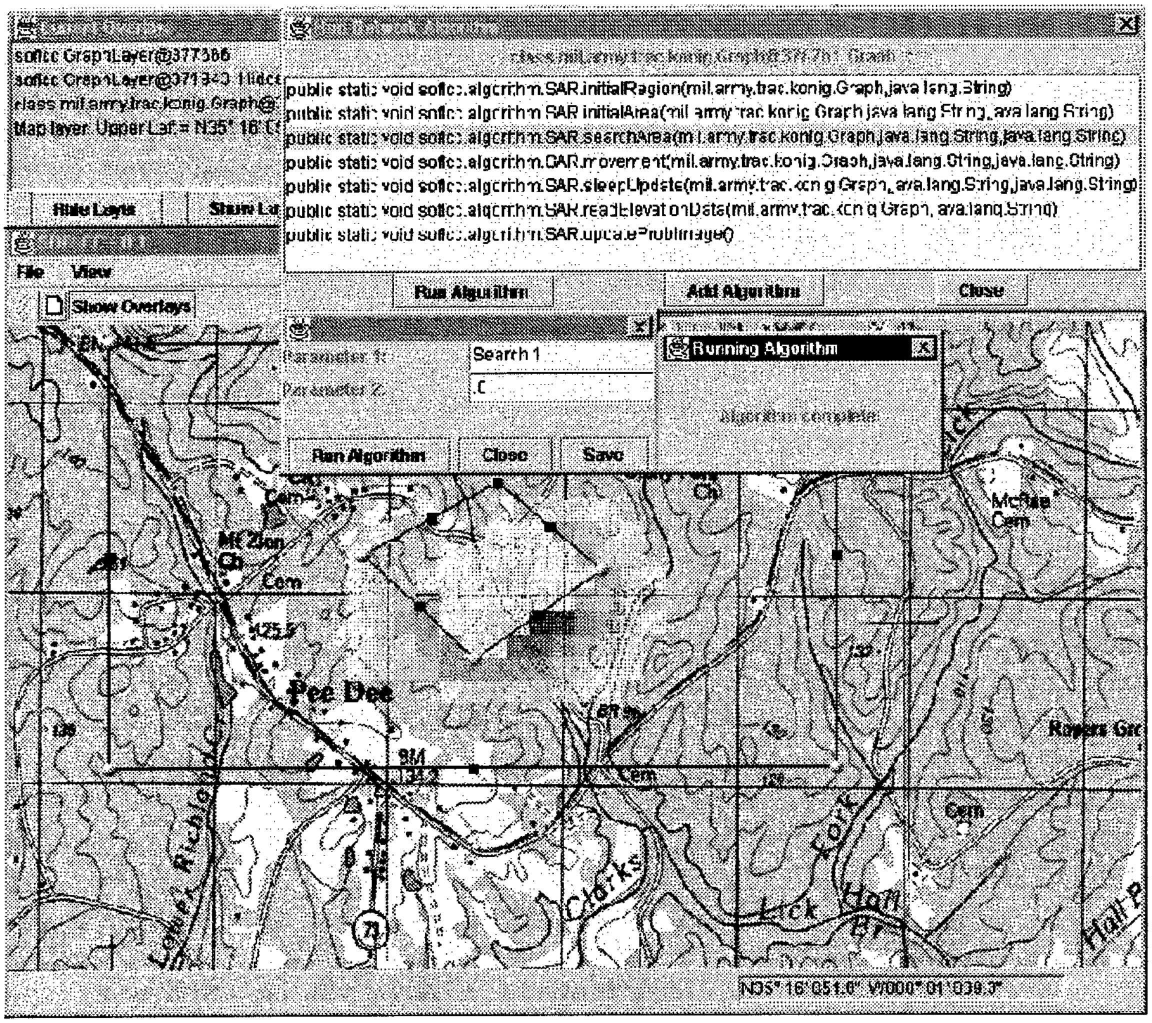
N35° 15' 024.4" W080° 01' 034.4"

Figure B-13. Use the Show Overlay icon to select the search region, and run the movement algorithm. The algorithm requires a name and a length of time in minutes. The probability map is updated based on the target movement that would occur during that time interval. A second movement option is available, the sleep update. This option also requires a name and a length of time in minutes. The time on the probability map is updated, but no target movement occurs, because the target is assumed to be sleeping.



N35° 16' 052.0" WC03° 01' 01 .0"

Figure B-14. Add a new overlay to create the search area. The area is created by single clicks of the mouse at each location of the map which is a vertex for the area. Double click at the last vertex to complete the area.



```
softc GraphLayer@377386
softc GraphLayer@371343 Hidden
class mil.army.trac.konig.Graph@
Map layer: Upper Lat = N35° 16' 0
```

```
class mil.army.trac.konig.Graph@37f7b1: Graph
public static void softc.algorithm.SAR.initialRegion(mil.army.trac.konig.Graph,java.lang.String)
public static void softc.algorithm.SAR.initialArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.searchArea(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.movement(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.sleepUpdate(mil.army.trac.konig.Graph,java.lang.String,java.lang.String)
public static void softc.algorithm.SAR.readElevationData(mil.army.trac.konig.Graph,java.lang.String)
public static void softc.algorithm.SAR.updateProbImage()
```

Parameter 1: Search 1

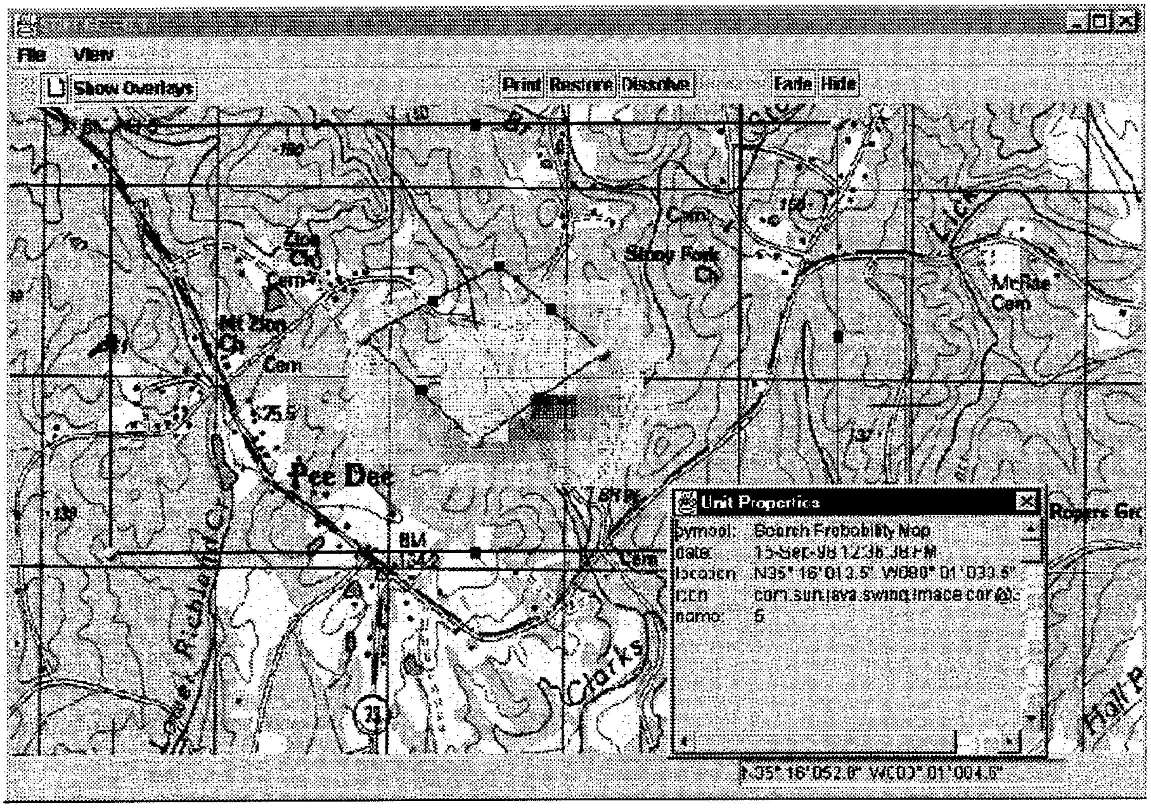
Parameter 2: .0

Running Algorithm

Algorithm complete

N35° 16' 051.6" W000° 01' 039.3"

Figure B-15. Use the Show Overlay icon to select the search area graph, and run the searchArea algorithm. This algorithm requires a name and a probability between 0 and 1 as parameters. The probability map is updated based on the search results.



symbol: Search Probability Map
date: 15-Sep-98 12:36:38 PM
location: N35° 16' 013.5" W080° 01' 033.5"
icon: com.sun.java.swing.ImageIcon@
name: 6

N35° 16' 052.0" W080° 01' 004.6"

Figure B-16. Double clicking on the probability map displays the current date and time of the search.

APPENDIX C. COINSS CODE

This appendix contains the code for SARhex, and SARgrid. In addition, it discusses the driver program necessary to interface COINSS with a map-based planning system. SARhex is the basic building block for COINSS. SARgrid contains all the methods which operate the model.

The COINSS software is connected to a map-based planning system through a driver program that invokes the methods in COINSS. The driver invokes these methods by obtaining the necessary information from the map-based planning system. The driver then converts the information into the correct parameter form to call the methods in COINSS. The driver then invokes the methods using the proper parameters, and provides the results back to the map-based system.

To initialize the search region, the driver instantiates an object "SearchRegion" of class SARgrid. The constructor requires the latitude and longitude of the northwest corner and the southeast corner, each being double variables with units degree. The constructor also requires the height and width of this region on the screen in pixels, each being int variables.

To specify the initial area(s) the method probInitialDatum is invoked on the object SearchRegion. The first parameter is a two-dimensional array of type int. Each column of the two-row array is a vertex of the polygon that defines one initial area. Each column is the location, in grid squares (x,y), of a point. The second parameter is a double variable that is the probability ($0.0<j \leq 1.0$) that the target started in this area.

To specify the search area(s), the method probInfoUpdate is invoked on the object SearchRegion. The parameters are the same as the method probInitialDatum, where the double variable is the probability the search team will find the target, given he is located in the given search area.

To specify movement, the method probTimeUpdate is invoked on the object SearchRegion. This method requires no parameters. It will update the probability map for movement of one time step, based upon the constant time step defined by SARgrid (currently 15 minutes).

To specify time passing during the night, the method updateTime is invoked on the object SearchRegion. The only parameter is a double variable. This double is the number of seconds that have elapsed while the target sleeps.

To present the probability map, the method getProbMapIcon is invoked on the object SearchRegion. This method requires no parameters. It returns the current probability map as an Icon variable.

```
/**
* Tim Castle
* SAR Model Search Hex piece
* Thesis Project
* Latest Revision Date: September 14, 1998
*
* Thesis Advisors: Professor Gordon Bradley
*                  Professor Alan Washburn
* Second Reader:   Professor James Eagle
*
**/

/**
 *    This class creates the basic building block for the SARgrid program
**/
package soflcc.algorithm;
import java.awt.*;

public class SARhex{

// instance variables

   private int maxVel;                //sets the range of the velocity
                                      //vector arrays

   private int velIndex;
                                      //the index used to adjust array
                                      //value to actual velocity
                                      //velocity = (array index)- velIndex

   private int xCoord;                //horizontal grid position
   private int yCoord;                //vertical grid position
   private double elevation;          //grid elevation

   private double prob[][];           //probability that target is in grid
```

```
                                    //moving with a certain velocity
                                    //velocity has x and y components
                                    //each ranging from -velIndex
                                    // to +velIndex feet/second
   private double tempProb[][];     //holder for the next updated prob


// constructors

   public SARhex(int xval, int yval, double elev, int vel) {

      maxVel = vel;
      velIndex = (maxVel-1)/2;

      xCoord = xval;                      //creates each pixel hex
      yCoord = yval;
      setElevation(elev);                 //with 0 elevation until read in

      prob = new double[maxVel][maxVel];
      for(int i=0;i<=maxVel-1;i++)
         for(int j=0;j<=maxVel-1;j++){
            setProb(i,j,0.0);       //and a 0 prob until assigned
         }

      tempProb = new double[maxVel][maxVel];
      resetTempProb();

   }

// instance methods

   public void resetTempProb(){

      for(int i=0;i<=maxVel-1;i++)
         for(int j=0;j<=maxVel-1;j++){
            setTempProb(i,j,0.0);
         }
   }

   public void incTempProb(int i, int j, double value){
      tempProb[i][j]+= value;
   }

   public void updateProb(){
      for(int i=0;i<=maxVel-1;i++)
         for(int j=0;j<=maxVel-1;j++){
            setProb(i,j,getTempProb(i,j));
         }
      resetTempProb();
   }

   public void adjustProb(double value){
      for(int i=0;i<=maxVel-1;i++)
         for(int j=0;j<=maxVel-1;j++){
            setProb(i,j,value*getProb(i,j));
         }
   }

// Getter/Setter methods

   public void setProb(int i, int j, double value){
      prob[i][j]= value;
   }

   public void setProb(double value){
      for(int i=velIndex;i<=velIndex+2;i++)
         for(int j=velIndex;j<=velIndex+2;j++){
            setProb(i,j,value/9);
         }
   }

   public double getProb(int i, int j){
      return(prob[i][j]);
   }
```

```
    public void setTempProb(int i, int j, double value){
       tempProb[i][j]= value;
    }

    public double getTempProb(int i, int j){
       return(tempProb[i][j]);
    }

    public double getHexProb(){
       double tempProb = 0;
       for(int i=0;i<=maxVel-1;i++)
          for(int j=0;j<=maxVel-1;j++){
             tempProb += getProb(i,j);
          }
       return(tempProb);
    }

    public void setElevation(double value){
       elevation=value;
    }

    public double getElevation(){
       return(elevation);
    }


 // toString method
    public String toString() {
       return "Coordinate: ("+xCoord+","+yCoord+")  Probability ="+getHexProb();
    }



 }
```

```
/**
* Tim Castle
* SAR Model Search Grid program
* Thesis Project
* Latest Revision Date: September 13, 1998
*
* Thesis Advisors: Professor Gordon Bradley
*                  Professor Alan Washburn
* Second Reader:   Professor James Eagle
*
**/

/**  This code was written by Tim Castle as the main
 *   program for the COINSS model.  It uses the SARhex
 *   as elements of the grid.
**/
package soflcc.algorithm;
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import mil.army.trac.konig.*;
import java.util.*;
import com.sun.java.swing.*;
//import ReadDTED1;

public class SARgrid{

// instance variables

   public static int maxVel=9;            //Dimension of velocity array
   public static int velIndex=(maxVel-1)/2;

   public static int timeStep=900;        //Time Step increment in seconds
   public static int Scale = 16;          //Dimension of each grid hex in pixels

   private int gridWidth;
   private int gridHeight;
private int targetType;           //Numeric value for the type of
                                  //target being searched for
                                  //1 - small children (1-6)
                                  //2 - children (7-12)
                                  //3 - despondents/walkaways
                                  //4 - hunters
                                  //5 - hikers/misc.

private SARhex square[][];
private int printArray[];         //Array used to print grid to screen

private double initProbFactor;    //Total initial datum probability
private double maxElevation;      //Max elevation of any square in grid
private double gridDist;          //Size of each square in the grid (in feet)
private double currentTime;       //Current clock time
private double moveMatrix[][];    //Markov Matrix which gives probabilities
                                  //of moving from one velocity state to
                                  //another at each time step update
private double outer[];           //Prob that target is outside the
                                  //search region in 1 of 4 directions
                                  //(0-N, 1-E, 2-S, 3-W)
private double ulLat;             //Latitude (in decimal) of upper left grid corner
private double ulLong;            //Longitude (in decimal) of upper left grid corner
private double lrLat;             //Latitude (in decimal) of lower right grid corner
private double lrLong;            //Longitude (in decimal) of lower right grid corner
private int vertPixels;           //Number of Pixels extending vertically across grid
private int horPixels;            //Number of Pixels extending horizontally across grid
private double incLat;            //The decimal degrees of Latitude that each
                                  //scaled hex grid covers
private double incLong;           //The decimal degrees of Longitude that each
                                  //scaled hex grid covers

private Date searchDate;          //Current date and time of the search, updated
                                  //as the search progresses

private short[][] cellDTED1;      //Array of elevation data for a 1 degree square
```

```
// constructors

/**  when created, SARgrid requires the latitude and longitude of the northwest corner
 *   (lat1,long1) and southeast corner (lat2,long2), as well as the number of pixels in
 *   the region, both vertically (pix1) and horizontally (pix2)
**/


   public SARgrid(double lat1, double long1, double lat2, double long2, int pix1, int pix2) {
      double tempElev;

      searchDate = new Date();

      setGridDimensions(lat1,long1,lat2,long2,pix1,pix2);

      cellDTED1 = new short[1201][1201];

      square = new SARhex[gridWidth][gridHeight];
      maxElevation = 0.0;

      for (int i=0;i<=gridWidth-1;i++)
         for (int j=0;j<=gridHeight-1;j++){
               square[i][j]=new SARhex(i,j,0.0,maxVel);
         }

      getTargetMoveData();

      initProbFactor = 0.0;

      printArray = new int[gridWidth*gridHeight*Scale*Scale+1];
      for (int i=0;i<=gridWidth*gridHeight*Scale*Scale;i++)
         printArray[i]=0;

      outer = new double[4];
      for (int i=0;i<=3;i++)
         outer[i]=0.0;

      cellDTED1 = null;
```

```
    }

// instance methods

/**
*
**/

   public void probInitialDatum(int[][] initialAreaArray, double prob){

      double totalProbRegion;        //total prob in the region
      int count;                     //total number of squares in initial area

// If initProbFactor>0, then rescale each square's probability, so that the
// total search region probability = 1

      if(initProbFactor>0.0)
         for(int x=0;x<=gridWidth-1;x++)
            for(int y=0;y<=gridHeight-1;y++)
               square[x][y].adjustProb(initProbFactor);

// Count the number of squares located in the initial area

      count=0;
      for(int x=0;x<=gridWidth-1;x++)
         for(int y=0;y<=gridHeight-1;y++){
            if(inPoly(initialAreaArray,x,y)){
               count++;
            }
         }

// and assign (prob/count) probability to each square in the initial area

   for(int x=0;x<=gridWidth-1;x++)
         for(int y=0;y<=gridHeight-1;y++){
               if(inPoly(initialAreaArray,x,y)){
                  square[x][y].setProb(prob/count);
            }
         }

// total the probability in the entire search region
// then normalize each square's probability, so that the
// total search region probability = 1

      initProbFactor += prob;
      for(int x=0;x<=gridWidth-1;x++)
         for(int y=0;y<=gridHeight-1;y++)
            square[x][y].adjustProb(1/initProbFactor);

   }

   public void probTimeUpdate(){

   double tempPortion[] = new double[4];
   double newI;
   double newJ;
   int newGridI;
   int newGridJ;
   double newX[] = new double[maxVel];
   double newY[] = new double[maxVel];


      for (int i=0;i<=gridWidth-1;i++)
         for (int j=0;j<=gridHeight-1;j++)
            for (int x=0;x<=maxVel-1;x++)
               for (int y=0;y<=maxVel-1;y++){
                  if(square[i][j].getProb(x,y)>0){

                  newI = i + (timeStep*(x-velIndex)/gridDist);
                  newJ = j + (timeStep*(y-velIndex)/gridDist);
                  newGridI = (int)newI;
                  newGridJ = (int)newJ;
                  newX = updateVector(x,square[i][j].getProb(x,y),moveMatrix,maxVel);
                  newY = updateVector(y,square[i][j].getProb(x,y),moveMatrix,maxVel);

                  if(newGridI<=0){incOuter(3,square[i][j].getProb(x,y));}
   else if(newGridI>=gridWidth-1){incOuter(1,square[i][j].getProb(x,y));}
      else if (newGridJ<=0){incOuter(0,square[i][j].getProb(x,y));}
         else if (newGridJ>=gridHeight-1){incOuter(2,square[i][j].getProb(x,y));}
else{
if(newI-newGridI>.5){
   if(newJ-newGridJ>.5){
      tempPortion[0]=((newI-newGridI)-.5)*(1.5-(newJ-newGridJ));
      tempPortion[1]=((newI-newGridI)-.5)*((newJ-newGridJ)-.5);
      tempPortion[2]=(1.5-(newI-newGridI))*((newJ-newGridJ)-.5);
      tempPortion[3]=(1.5-(newI-newGridI))*(1.5-(newJ-newGridJ));
      for(int a=0;a<=maxVel-1;a++)
         for (int b=0;b<=maxVel-1;b++){
            square[newGridI+1][newGridJ].incTempProb(a,b,tempPortion[0]*newX[a]*newY[b]);
            square[newGridI+1][newGridJ+1].incTempProb(a,b,tempPortion[1]*newX[a]*newY[b]);
            square[newGridI][newGridJ+1].incTempProb(a,b,tempPortion[2]*newX[a]*newY[b]);
            square[newGridI][newGridJ].incTempProb(a,b,tempPortion[3]*newX[a]*newY[b]);
         }
   }
   else{
      tempPortion[0]=((newI-newGridI)-.5)*(.5+(newJ-newGridJ));
      tempPortion[1]=((newI-newGridI)-.5)*(.5-(newJ-newGridJ));
      tempPortion[2]=(1.5-(newI-newGridI))*(.5-(newJ-newGridJ));
      tempPortion[3]=(1.5-(newI-newGridI))*(.5+(newJ-newGridJ));
      for(int a=0;a<=maxVel-1;a++)
         for (int b=0;b<=maxVel-1;b++){
            square[newGridI+1][newGridJ].incTempProb(a,b,tempPortion[0]*newX[a]*newY[b]);
            square[newGridI+1][newGridJ-1].incTempProb(a,b,tempPortion[1]*newX[a]*newY[b]);
            square[newGridI][newGridJ-1].incTempProb(a,b,tempPortion[2]*newX[a]*newY[b]);
            square[newGridI][newGridJ].incTempProb(a,b,tempPortion[3]*newX[a]*newY[b]);
         }
   }
}
else{
   if(newJ-newGridJ>.5){
      tempPortion[0]=(.5-(newI-newGridI))*(1.5-(newJ-newGridJ));
      tempPortion[1]=(.5-(newI-newGridI))*((newJ-newGridJ)-.5);
      tempPortion[2]=(.5+(newI-newGridI))*((newJ-newGridJ)-.5);
      tempPortion[3]=(.5+(newI-newGridI))*(1.5-(newJ-newGridJ));
```

```
                     for(int a=0;a<=maxVel-1;a++)
                        for (int b=0;b<=maxVel-1;b++){
                           square[newGridI-1][newGridJ].incTempProb(a,b,tempPortion[0]*newX[a]*newY[b]);
                           square[newGridI-1][newGridJ+1].incTempProb(a,b,tempPortion[1]*newX[a]*newY[b]);
                           square[newGridI][newGridJ+1].incTempProb(a,b,tempPortion[2]*newX[a]*newY[b]);
                           square[newGridI][newGridJ].incTempProb(a,b,tempPortion[3]*newX[a]*newY[b]);
                        }
                  }
                  else{
                     tempPortion[0]=(.5-(newI-newGridI))*(.5+(newJ-newGridJ));
                     tempPortion[1]=(.5-(newI-newGridI))*(.5-(newJ-newGridJ));
                     tempPortion[2]=(.5+(newI-newGridI))*(.5-(newJ-newGridJ));
                     tempPortion[3]=(.5+(newI-newGridI))*(.5+(newJ-newGridJ));
                     for(int a=0;a<=maxVel-1;a++)
                        for (int b=0;b<=maxVel-1;b++){
                           square[newGridI-1][newGridJ].incTempProb(a,b,tempPortion[0]*newX[a]*newY[b]);
                           square[newGridI-1][newGridJ-1].incTempProb(a,b,tempPortion[1]*newX[a]*newY[b]);
                           square[newGridI][newGridJ-1].incTempProb(a,b,tempPortion[2]*newX[a]*newY[b]);
                           square[newGridI][newGridJ].incTempProb(a,b,tempPortion[3]*newX[a]*newY[b]);
                        }
                  }
               }
               }
            }
         }


   for (int i=0;i<=gridWidth-1;i++)
      for (int j=0;j<=gridHeight-1;j++){
            square[i][j].updateProb();
         adjustForElevation(i,j);
      }

}

public void adjustForElevation(int i, int j){

   double newI;
double newJ;
int newGridI;
int newGridJ;
double squareProbStart;
double squareProbEnd;

squareProbStart=square[i][j].getHexProb();

for (int x=0;x<=maxVel-1;x++)
   for (int y=0;y<=maxVel-1;y++){
      if(square[i][j].getProb(x,y)>0){

         newI = i + (timeStep*(x-velIndex)/gridDist);
         newJ = j + (timeStep*(y-velIndex)/gridDist);
         newGridI = (int)newI;
         newGridJ = (int)newJ;

         if(newGridI<0){newGridI=0;}
            else if(newGridI>gridWidth-1){newGridI=gridWidth-1;}
         if(newGridJ<0){newGridJ=0;}
            else if(newGridJ>gridHeight-1){newGridJ=gridHeight-1;}
         switch(targetType){
            case 1: if(Math.abs(slope(i,j,newGridI,newGridJ))>.3){
                        square[i][j].setProb(x,y,0.0);}
                  break;
            case 2: if(Math.abs(slope(i,j,newGridI,newGridJ))>.45){
                        square[i][j].setProb(x,y,0.0);}
                        else if(Math.abs(slope(i,j,newGridI,newGridJ))>.3){
                              square[i][j].setProb(x,y,square[i][j].getProb(x,y)/2);}
                  break;
            case 3: if(Math.abs(slope(i,j,newGridI,newGridJ))>.45){
                        square[i][j].setProb(x,y,0.0);}
                        else if(Math.abs(slope(i,j,newGridI,newGridJ))>.3){
                              square[i][j].setProb(x,y,square[i][j].getProb(x,y)/2);}
                  break;
            case 4: if(Math.abs(slope(i,j,newGridI,newGridJ))>.7){
                        square[i][j].setProb(x,y,0.0);}
                        else if(Math.abs(slope(i,j,newGridI,newGridJ))>.45){
```

```
                                 square[i][j].setProb(x,y,square[i][j].getProb(x,y)/2);}
                     break;
               case 5: if(Math.abs(slope(i,j,newGridI,newGridJ))>.7){
                           square[i][j].setProb(x,y,0.0);}
                           else if(Math.abs(slope(i,j,newGridI,newGridJ))>.45){
                                 square[i][j].setProb(x,y,square[i][j].getProb(x,y)/2);}
                     break;
               }
            }
         }
   squareProbEnd=square[i][j].getHexProb();
   for (int x=0;x<=maxVel-1;x++)
      for (int y=0;y<=maxVel-1;y++){
         if(square[i][j].getProb(x,y)>0){
            square[i][j].setProb(x,y,square[i][j].getProb(x,y)*squareProbStart/squareProbEnd);
         }
      }

}

public double slope(int i, int j, int nexti, int nextj){

   double rise = square[i][j].getElevation()-square[nexti][nextj].getElevation();
   double run = gridDist*(Math.sqrt((i-nexti)*(i-nexti)+(j-nextj)*(j-nextj)));
   return(rise/run);
}

public double[] updateVector(int y, double prob, double[][] matrix, int length){

   double row[] = new double[length];

   for(int i=0;i<=length-1;i++){
      row[i]=Math.sqrt(prob)*matrix[y][i];
   }
   return(row);
}

public void probInfoUpdate(int[][] searchAreaArray, double pDetect){
      double totalProbStart;          //total prob of target in search region
      double totalProbEnd;            //prior to update, and after update

      totalProbStart = 1-outer[0]-outer[1]-outer[2]-outer[3];
      totalProbEnd = 0;

// If a grid hex is located in the search update region
// adjust the likelihood probability by multiplying times
// the probability the search team would not have detected
// the target (1-pDetect)

      for(int x=0;x<=gridWidth-1;x++)
         for(int y=0;y<=gridHeight-1;y++){
            if(inPoly(searchAreaArray,x,y)){
               square[x][y].adjustProb(1-pDetect);
            }
         }

// The previous loop modified the likelihood probabilities
// to decrease the likelihood in searched regions.  Likewise,
// the likelihood in non-searched regions must increase.  In order
// to update for this, as well as maintain a total likelihood = 1,
// we must update the entire grid with a consistent ratio.

      for(int x=0;x<=gridWidth-1;x++)
         for(int y=0;y<=gridHeight-1;y++){
            totalProbEnd += square[x][y].getHexProb();
         }

      for(int x=0;x<=gridWidth-1;x++)
         for(int y=0;y<=gridHeight-1;y++){
            square[x][y].adjustProb(totalProbStart/totalProbEnd);
         }
   }

/**   Modified code from Inpoly.C
 *    Copyright (c) 1995-1996 Galacticomm, Inc.  Freeware source code.
 *    Home for this file:  http://www.gcomm.com/develop/inpoly.c
 *
 *                                  6/19/95 - Bob Stein & Craig Yap
 *                                  stein@gcomm.com
 *                                  craig@cse.fau.edu
 *
 *                poly is an array of integers with the first value 0 or 1
 *                0 - x, 1 - y, and the second value denotes the position
 *                in the array 0 .. length-1
**/

   public boolean inPoly(int[][] poly, int xt, int yt){

   int xnew;
   int ynew;
   int xold;
   int yold;
   int x1;
   int x2;
   int y1;
   int y2;
   boolean inside;
   int npoints;

      inside = (0>1);
      npoints = poly[0].length;

      if (npoints >= 3) {
         xold=poly[0][npoints-1];
         yold=poly[1][npoints-1];
         for (int i=0 ; i < npoints ; i++) {
          xnew=poly[0][i];
          ynew=poly[1][i];
          if (xnew > xold) {
               x1=xold;
               x2=xnew;
               y1=yold;
               y2=ynew;
```

```
            }
            else {
                 x1=xnew;
                 x2=xold;
                 y1=ynew;
                 y2=yold;
            }
            if ((xnew < xt) == (xt <= xold)          /* edge "open" at left end */
             && ((long)yt-(long)y1)*(long)(x2-x1)
              < ((long)y2-(long)y1)*(long)(xt-x1)) {
                 inside=!inside;
            }
            xold=xnew;
            yold=ynew;
           }
      }
      return(inside);
}

   public void incOuter(int place, double value){
      outer[place]+=value;
   }


/**
 *This code was taken from the Marine Navigation Calculator page
 * of the Marine Navigation Department at NIMA
 * This site is located at
 * http://164.214.2.59/Navigation/bowditch/calc/tab-intro_nfr.html
 * The code was developed from the tables at the back of
 * The American Practical Navigator
**/
   public double computeDistance(double value, double longDist){

      double p1 = 111412.84;         // longitude calculation term 1
      double p2 = -93.5;             // longitude calculation term 2
      double p3 = 0.118;             // longitude calculation term 3
                                     // convert the given lat from
```

```
                                    // degrees to radians
      double lat = (value* 2.0 * Math.PI)/360.0;

//This equation takes a particular degree of latitude,
//and computes the distance, in meters, of 1 degree of longitude

      double longlen = (p1 * Math.cos(lat)) + (p2 * Math.cos(3 * lat)) +
                                        (p3 * Math.cos(5 * lat));
System.out.println(longlen+" at the latitude "+value+" and Distance "+longDist);
//I wanted to return distance in feet, so I converted the answer
//before returning the value

      return(longDist*longlen*39.370079/12);
   }

//This method assigns a color to the hex, based on its
//likelihood value.  The maxSquareProb is assigned the purest
//color of red.  Every 10% loss in likelihood results in a
//10% loss in the shade of red (based on the RGB value).
//The loss continues until the lowest likelihood hexes are
//shaded almost completely white.

   public int computeProbColor(int i, int j, double max){
      int thisColor;

      double value = square[i][j].getHexProb();
      if(value<(0.1*max)){
         thisColor = ((0)<<24|255<<16|255<<8|255);}
      else{
         int base = (255-25*(int)(10*value/max));
         thisColor = ((190)<<24|255<<16|base<<8|base);
      }
      return(thisColor);
   }

   public int computeElevColor(int i, int j){

      double value = square[i][j].getElevation();
      int base = (255-25*(int)(10*value/maxElevation));
      return((255)<<24|base<<16|base<<8|255);
   }


//This method creates an array of colors, which correspond to the
//likelihood probability of each hex.  The shades of colors assigned
//are computed in the computeProbColor method.

   public void setPrintArray(){
      int value;
   double maxSquareProb = 0.0;

   for(int j=0;j<=gridHeight-1;j++)
         for(int i=0;i<=gridWidth-1;i++)
          maxSquareProb=Math.max(maxSquareProb,square[i][j].getHexProb());

      for(int j=0;j<=gridHeight-1;j++)
         for(int i=0;i<=gridWidth-1;i++){
            value = computeProbColor(i,j,maxSquareProb);
            for(int x=Scale*j;x<=Scale*j+Scale-1;x++)
               for(int y=Scale*i;y<=Scale*i+Scale-1;y++)
                  printArray[x*Scale*gridWidth+y] = value;
         }
   }

   public int[] getPrintArray(){
      setPrintArray();
      return(printArray);
   }

   public int[] getPrintElev(){
      int value;
      for(int j=0;j<=gridHeight-1;j++)
         for(int i=0;i<=gridWidth-1;i++){
            value = computeElevColor(i,j);
            for(int x=Scale*j;x<=Scale*j+Scale-1;x++)
               for(int y=Scale*i;y<=Scale*i+Scale-1;y++)
               printArray[x*Scale*gridWidth+y] = value;
      }
   return(printArray);
}

public Icon getProbMapIcon(){
int width=gridWidth*Scale;
   int height=gridHeight*Scale;
   int[] pixels = new int[width*height];
   pixels = getPrintArray();

ImageProducer ip =
    new MemoryImageSource(width, height, pixels, 0, width);
 Icon icon = new ImageIcon(Toolkit.getDefaultToolkit().createImage(ip));
 return(icon);
 }

public void outPrintArray(){
   for(int i=0;i<=gridHeight*gridWidth;i++)
      System.out.println("Location: "+i+"  Color:  "+printArray[i]);
}

public void readMoveFile( String fileName)  throws IOException  {

   DataInputStream in = new DataInputStream(new FileInputStream( fileName ));

   // The first item in the file is an integer identifying the target type

   targetType = in.readInt();

   // moveMatrix is [maxVel x maxVel] array of doubles
   // representing a Markov chain
   // each row represents the probability of changing states from the
   // current velocity to a new velocity
   // This state change occurs at each movement update
   //

   moveMatrix = new double[maxVel][maxVel];
```

```
         for (int i=0;i<=maxVel-1;i++){
            for (int j=0;j<=maxVel-1;j++){
               moveMatrix[i][j]=in.readDouble();
            }
         }

         in.close();
      }

      public void updateElevation(){
      double tempElev;

      getElevationData();

      maxElevation = 0.0;

      for(int i=0;i<=gridWidth-1;i++)
         for(int j=0;j<=gridHeight-1;j++){
            tempElev = getElevation((ulLong+(i+0.5)*incLong),(ulLat-(j+0.5)*incLat));
            maxElevation = Math.max(tempElev,maxElevation);
            square[i][j].setElevation(tempElev);
         }
      }

      public void updateTime(double seconds){
         long tempMilli;

         tempMilli = searchDate.getTime();
         tempMilli += seconds*1000;
         searchDate.setTime(tempMilli);
      }

   // Getter/Setter methods

   public void getTargetMoveData(){

         Frame frame = new Frame();
         FileDialog fileDialog = new FileDialog(frame, "Choose Target Move file");
      String fileName;

      fileDialog.setMode(FileDialog.LOAD);
      fileDialog.show();
      fileName = fileDialog.getFile();
      if (fileName == null) {
         fileName = "dted/w098/n31.dt1";
      }
      else {
         fileName = fileDialog.getDirectory() + fileName;
      }
      try {
          readMoveFile(fileName);
      }
      catch(IOException e)  {System.out.println(e);}
   }


   public void setGridDimensions(double lat1, double long1, double lat2, double long2, int pix1, int pix2){

      ulLat = lat1;           //reads in values provided
      ulLong = long1;
      lrLat = lat2;
      lrLong = long2;
      vertPixels = pix1;
      horPixels = pix2;

      incLat = ((ulLat-lrLat)*Scale)/vertPixels;      //calculates the size of each
      incLong = ((lrLong-ulLong)*Scale)/horPixels;    //scaled hex in decimal degrees
                                                      //of Lat and Long

      gridHeight = ((vertPixels-1)/Scale)+1;          //Calculates the number of scaled
      gridWidth = ((horPixels-1)/Scale)+1;            //hexes required to completely
                                                //cover the search map area
System.out.print(gridHeight+"x"+gridWidth);

      vertPixels = gridHeight*Scale;            //Recalculates the number of pixels
      horPixels = gridWidth*Scale;              //the scaled search map encompasses and
      lrLat = ulLat-incLat*gridHeight;          //assigns a new lower right Lat/Long
      lrLong = ulLong+incLong*gridWidth;        //based on this new scaled coverage map

                                                //computes the distance in feet of each
                                                //scaled grid hex, based on the current latitude

      gridDist = computeDistance((ulLat+lrLat)/2,lrLong-ulLong)/gridWidth;


   }

//This method was taken from the main method of class ReadDTED1
//writen by professor Gordon Bradley

   public void getElevationData(){

      Frame frame = new Frame();
      FileDialog fileDialog = new FileDialog(frame, "Choose DTED file");
      String fileName;

      fileDialog.setMode(FileDialog.LOAD);
      fileDialog.show();
      fileName = fileDialog.getFile();
      if (fileName == null) {
         fileName = "dted/w098/n31.dt1";
      }
      else {
         fileName = fileDialog.getDirectory() + fileName;
      }
      try {
          ReadDTED1.readDTED1File(fileName,cellDTED1);
      }
      catch(IOException e)  {System.out.println(e);}
   }

/**
*     This getter returns the elevation at a given longitude/latitude
*     The method assumes that the long/lat provided as (i,j) are within
*     the same degree of long/lat selected during the readDTED1 method
**/

   public double getElevation(double i, double j){

   i=Math.abs(i);
   j=Math.abs(j);
      return(cellDTED1[(int)(1200-((i-(int)i)*1200))][(int)((j-(int)j)*1200)]);
   }



   public SARhex getSquare(int first, int second){
      return(square[first][second]);
   }

   public int getXPixelValue(double val){
      int xPixel;

      if(val<ulLong){
         xPixel = 0;
      }
      else if(val>lrLong){
            xPixel = gridWidth;
            }
            else{
               xPixel = (int)(Math.round((val-ulLong)/incLong));
            }

      return(xPixel);
   }

   public int getYPixelValue(double val){
      int yPixel;

      if(val>ulLat){
         yPixel = 0;
      }
      else if (val<lrLat){
            yPixel = gridHeight;
            }
            else{
               yPixel = (int)(Math.round((ulLat-val)/incLat));
            }

      return(yPixel);
   }

   public int getGridHeight(){

   return(gridHeight);
   }

   public int getGridWidth(){

   return(gridWidth);
   }

   public String presentSearchDate(){

      return(searchDate.toLocaleString());
   }


   // toString method



}
```

LIST OF REFERENCES

[1] Benkoski, S.J., Monticino, M.G., and Weisinger, J.R., *A Survey of the Search Theory Literature*, Naval Research Logistics, **38**, 469-494 (1991).

[2] Bilyeu, A.L., *Concept for a Special Operations Planning and Analysis System*, Master's Thesis in Operations Research, Naval Postgraduate School, Monterey, California, 1998.

[3] Bownds, J.M., Ebersole, M., and Lovelock, D., *Computer Aided Search Information Exchange III (CASIE III)*, 1992.

[4] Bradley, G.H. and Buss, A.H., *An Architecture for Dynamic Planning Systems Using Loosely Coupled Components*. Technical Report, Naval Postgraduate School, Monterey, California, 1998.

[5] Cornell, G. and Horstmann, C.S., *Core Java*, Sunsoft Press, 1996.

[6] Davison, M., *Surface Search and Localization Tactical Decision Aid (Surface SALT)*. Unpublished manuscript, 1987.

[7] Flanagan, D., *Java in a Nutshell*, O'Reilly & Associates, 1997.

[8] Geary, D., *Graphic Java 1.1*, Sun Microsystems Press, 1997.

[9] Hill, K., *Predicting the Behavior of Lost Persons*. Waverly Ground Search and Rescue and Saint Mary's University, Canada.

[10] National Imagery & Mapping Agency, *Marine Navigation Calculator*, http://164.214.2.59/Navigation/bowditch/calc/degree.html, 1997.

[11] Recalde, C.J., *A Reactive Target Active ASW Sonar Search Tactical Decision Aid*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1996.

[12] Recalde, C.J., *A Search and Rescue Decision Aid Implementation in Java*. Unpublished manuscript, Department of Operations Research, Naval Postgraduate School, Monterey, California, 1996.

[13] Ross, S.M., *Introduction to Probability Models*, Academic Press Inc., 1993.

[14] Stein, B., *A Point about Polygons*, http://www.gcomm.com/develop/inpoly.txt, 1997.

[15] Stein, B. and Yap, C., *Inpoly.C*, http://www.gcomm.com/develop/inpoly.c, 1995.

[16] Stone, L.D., *The Process of Search Planning: Current Approaches and Continuing Problems*, Operations Research, **31**, 207-233 (1983).

[17] Syrotuck, W.G., *Analysis of Lost Person Behavior: An Aid to Search Planning*, Arner Publications, 1977.

[18] U.S. Coast Guard, Joint Chiefs of Staff, *National Search and Rescue Manual*, 1991.

[19] Wagner, D.H., *Naval Tactical Decision Aids*. Military Operations Research lecture notes, Naval Postgraduate School, Monterey, California, 1989.

[20] Washburn, A.R., *Search and Detection*, Institute for Operations Research and the Management Sciences, 1996.

[21] Washburn, A.R., *On Search for a Moving Target*, Naval Logistics Quarterly, **27**, 315-322 (1980).

[22] Washburn, A.R., *A Short Introduction to Kalman Filters*, Unpublished Text, Operations Research Department, Naval Postgraduate School, Monterey, California, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Professor Gordon Bradley, Code OR/Bz 2
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Professor Alan Washburn, Code OR/Ws 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Professor James Eagle, Code OR/Er 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. LT Timothy Castle 1
   450 Hardmoore Ct.
   Glen Burnie, MD 21061-6100

7. Commandant(G-WR-2) 1
   U.S. Coast Guard
   2100 Second Street S.W.
   Washington, D.C. 20593-0001
   Attn: LCDR Lehocky

8. Commandant(G-LGL) 1
   U.S. Coast Guard
   2100 Second Street S.W.
   Washington, D.C. 20593-000

9. USCG Research and Development Center 1
   1082 Shennecossett Rd.
   Groton, CT 06340-6096
   Attn: Kevin Downer

10. National Search and Rescue School 1
USCG Reserve Training Center
Yorktown, VA 23690-5000
Attn: LtCol Steve Roark

11. David Lovelock 1
Department of Mathematics
University of Arizona
Tuscon, AZ 85721

12. Dr. Anthony S. G. Jones 1
Marine Science Labs
Menai Bridge
Gwynedd LL59 5EY
Wales, UK

13. Robert Stoffel 1
Emergency Response Institute
Cashmere, WA 98815